

Improving Technologies for the Manipulation and Measurement of Brain Activity

by

Michael Alan Moncrieff Henninger

B.A. Physics, Math, and Integrated Science
Northwestern University, 2003



SUBMITTED TO THE DEPARTMENT OF PHYSICS IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY IN PHYSICS
AT THE
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

~~JUNE 2016~~ [June 2017]

© 2017 Massachusetts Institute of Technology. All rights reserved

Signature redacted

Signature of Author: _____

Signature redacted Department of Physics
May 26, 2017

Certified by: _____

Signature redacted
Ed Boyden
Professor, Media Lab and McGovern Institute,
Departments of Biological Engineering and Brain and Cognitive Science
Thesis supervisor

Signature redacted

Ibrahim Cissé
Assistant Professor of Physics
Thesis co-supervisor

Signature redacted

Accepted by: _____

Nergis Mavalvala
Professor of physics, Associate Department Head for Education

Improving Technologies for the Manipulation and Measurement of Brain Activity

by

Michael Alan Moncrieff Henninger

Submitted to the Department of Physics on May 26, 2017
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Physics

ABSTRACT

The ability to measure and manipulate brain activity is integral to much of neuroscience. Several modalities can be used to transmit information between the brain tissue and the experimental device, each with its own benefits and drawbacks. In this thesis, I treat two major thrusts towards emerging technologies that improve upon optical information transmission modalities.

I examine light propagation in the brain, with a focus on its importance to optogenetic manipulation of neural activity. Optogenetics is emerging as a powerful tool for neural activity manipulation: it is fast, can be genetically targeted to specific cell types, and can provide bidirectional control (inhibition or stimulation). Manipulation requires transmitting light from an experimental device to the light sensitive proteins that modulate the cell's activity. The brain is a highly scattering, highly absorbing, non-homogeneous medium. I created a Monte Carlo code to simulate light's propagation through this medium and used it to guide the design of optogenetic stimulators and predict the *in vivo* performance of new optogenetic proteins.

I designed and computationally evaluated the performance of a new kind of imager—referred to as an Implantable Light Field Microimager (ILM)—when used to measure neural activity reported by genetically encoded calcium sensors. Fluorescent reporters of physiological activity are already important tools in the study of neural dynamics, but recording the optical signals with sufficient temporal and spatial resolution—especially in deep brain structures—remains challenging due to the optical properties of brain tissue. The ILM fuses recent developments in light field imaging and computational photography with an algorithm that uses priors to solve the otherwise-underconstrained reconstruction problem. My simulations indicate that such a device would be effective at achieving single-cell resolution of neural activity at high speeds, with minimal tissue displacement and impact on brain temperatures—an often overlooked aspect of brain implants that can have major impacts on neural activity.

Thesis Supervisor: Ed Boyden

Title: Professor, Media Lab and McGovern Institute, Departments of Biological Engineering and Brain and Cognitive Science

I would like to acknowledge the loving support of my wife Abby, who has been there always, through wonderful times and so many hardships on this journey.

I also thank my children, Beth and Ricky, for being my refreshing inspirations/unwitting accomplices. You are the brightest lights in my world.

Thanks to my parents for being supportive every step of the way over the many years, and for all their help for me and my family during the trials and tribulations of the last few years.

I would like to thank my advisor, Ed Boyden, for taking a chance on me—then sticking with me as the first attempt(s) didn't succeed. I am certain that this thesis is only the first part of the payoff.

Thanks to each and every member of the physics PhD entering class of 2006; your support and patient assistance was immeasurably helpful in getting back into advanced physics after my time in the Peace Corps.

Thanks also to all the members—past and present—of the Boyden lab, as well as the Media Lab affiliates and Raskar lab collaborators with whom I've worked. Trading ideas and knowledge with you is the stuff that makes being a scientist such a joy.

To the people of cookie Monday, your comradery and stimulating conversation are always missed.

Thanks to Vicky Kalogera, my undergrad advisor... You are one of my biggest role models as a scientist and mentor.

With deep gratitude, I remember Bob Gideon, with whom it all began.

Contents

I.	Simulating light delivery in brains	6
A.	Existing fluence measurements and calculations	6
B.	Monte Carlo simulations of light propagation and fluence in vivo	11
1.	Motivations for selecting a Monte Carlo model	11
2.	Inputs and assumptions for the model	12
3.	Choice of tissue optical parameters.....	17
4.	Algorithm	23
5.	Implementation	26
C.	In vivo measurements of light fluence and heating	32
D.	Results.....	39
1.	Contributions to “Prosthetic systems for therapeutic optical activation and silencing of genetically targeted neurons”	39
2.	Contributions to “Characterization of the functional MRI response temporal linearity via optical control of neocortical pyramidal neurons”	43
3.	Contributions to “High-performance genetically targetable optical neural silencing by light-driven proton pumps”	44
4.	Contributions to “Noninvasive optical inhibition with a red-shifted microbial rhodopsin”	45
II.	Implantable light field microimager: designing and evaluating a new brain imager	48
A.	Introduction	48
1.	Device architecture	49
2.	Principles of device operation	50
B.	Results.....	52
1.	Simulated device performance and performance limits	52
2.	Temperature changes caused by the device	57
3.	Sensitivity to input parameters.....	60
C.	Methods.....	61
1.	Simulating light propagation in brain tissue	62
2.	Simulating neuronal activity and resulting fluorescence.....	62
3.	Simulating ILM images	63
4.	Reconstructing physiological activity.....	65
5.	Simulating Heating.....	67
D.	Discussion.....	68

E.	Future directions.....	70
1.	Building an ILM.....	70
2.	A first ILM experiment	71
3.	Excitation sources and their use in improving ILM performance	72
4.	Improved mathematical frameworks	73
5.	Imager improvements.....	75
6.	Brain temperature changes induced by implants and craniotomies.....	76
III.	Appendices.....	85
A.	Photon propagation Monte Carlo code	85
B.	Implantable light field microimager simulation code.....	104

I. Simulating light delivery in brains

Optogenetic tools are revolutionizing many aspects of neuroscience¹⁻⁸ and are in use in literally thousands of labs⁹. The ability to activate or inhibit brain cells with millisecond precision in a spatially- and genetically- targeted way opens up entire new categories of neuroscience experiments. In optogenetics, the gene for a light sensitive ion channel or pump (herein referred to generally as “opsin” for simplicity) is introduced to cells with the desired promoter to enable expression in the cell type(s) of interest¹⁰. Then, during the experiment, excitation light is delivered to those cells to generate the desired stimulation pattern. A key variable in such experiments is the excitation light fluence. Fluence is a quantity closely related to irradiance, but whereas irradiance is light power per unit area normal to a surface, fluence does not depend on direction of the light and is the “gross flux” of photons passing through a volume. The sensitivity of opsins in cells is effectively isotropic, and the excited current is dependent only on the number of excitation photons passing through the volume containing the opsins, not on their direction, making fluence the appropriate quantity to consider. Each opsin has a fluence-current response curve indicating how much current (for a particular set of intra- and extra-cellular conditions) an opsin molecule will deliver under a given photon fluence. These curves are straightforward to measure *in vitro*, and are typically included in any paper introducing a new opsin. See Figure 1 for an example. To understand the electrical excitation a cell receives via the optogenetic stimulation, one must know both the fluence-current curve and the fluence itself.

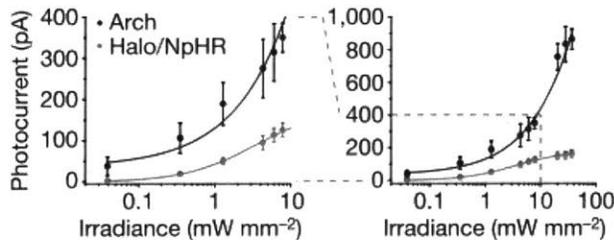


Figure 1 A typical irradiance-fluence plot for two opsins. In this case (collimated light illuminating a single-cell sample), the fluence is equal to the irradiance. From Chow, Han, et al 2010

For an experiment performed with either cultured cells or a thin brain slice in a microscope, determining the excitation fluence is relatively straightforward: The total excitation power delivered can be measured by, e.g., a photometer with an integrating sphere, and the illumination pattern is known—often design to be a nearly uniform, collimated beam in the field of view. This allows for simple calculation of the excitation irradiance. However, for an *in vivo* experiment, no such simple measurements or

calculations exists. Light is often delivered by optical fiber or other wave guide, in which case the excitation light is nonuniform in both position and angular distribution. Moreover, the light is entering a volume of tissue that is—even for a mouse brain—much larger than the absorption and effective isotropic scattering lengths. The fluence is thus a function that varies rapidly with position in the brain.

A. Existing fluence measurements and calculations

Simple calculators and experiments based on brain slices of varying thicknesses in microscopes have been presented in relation to optogenetic light delivery¹¹⁻¹³, but several factors limit their accuracy: confounding geometric factors of the measured system, oversimplified formulae used in calculators, tissue properties in the experiments deviating from their *in vivo* values, and so on.

Aravansis et al.¹¹ produced one of the first fluence calculations in the context of optogenetics. They aligned an optical fiber over a slices of brain tissue of varying thickness and recorded the total light

power transmitted with a photodetector beneath the brain slice, taking the incident power to be the measured power when no brain slice was present. They then fit the results to the zero absorption version of the Kubelka-Munk equation:

$$\frac{I(z)}{I(0)} = \frac{1}{Sz + 1}$$

Where $I(z)$ is intensity at a distance z into the scattering medium, and S is the scattering coefficient. In mouse with light of wavelength 473nm they measured S to be 11.2 mm^{-1} . The form of Kubelka-Munk (K-M) equation used by Aravansis assumes the tissue is a planar, homogeneous, non-absorptive, ideal diffuser; and that the excitation light is planar, diffuse and collimated. All of these assumptions frequently break down in or near the regime relevant for optogenetics, especially in vivo.

In an in vivo experiment, the light source is often implanted inside the brain (though see I.D.4 for an important new class of exceptions), placed within a few hundred microns of the brain region to be stimulated, due to the rapid falloff in light power (see, e.g., the 532nm light in Figure 15 and Figure 16). The tissue thus extends around and behind the light source. The illuminated area of the brain is also bounded by tissues with different optical properties, such as cerebrospinal fluid, bone, white matter tracts (or grey matter, if illuminating a white matter tract), etc. These boundaries—while often well-defined and sharp—are irregular and do not fit the planar and homogeneous assumptions of the Kubelka-Munk equation. That said, if the distance to those boundaries is substantially greater than the light penetration depth, the deviation from the planar assumption may not have a meaningful impact.

Another source of inhomogeneity in the tissue is the microscale variability of optical properties within a segment of tissue. For example, a neuronal cell body will have different scattering and absorption coefficients than its dendritic arbor. This brings up a serious question about coarse-graining and using these bulk scattering and absorption coefficients: Indeed, the large majority of scattering in tissue is Mie scattering, where the light refracts from one cellular compartment into another¹⁴; On a sufficiently small scale, the scattering coefficient is nearly zero and photons change direction due to refraction. Such a simulation of refraction through all cellular compartments is both computationally intractable and would require submicron resolution morphological data about the particular tissue to be simulated. That said, there are reasonably good arguments that such a fine-grained treatment is not required: typical light source dimensions and the light penetration depth are both on the order of at least tens of microns, so averaging over those scales is unlikely to be problematic. Also, the variations on the micron scales are typically not especially large; mostly refractive index changes of relatively small magnitude and/or material properties that vary on strongly subwavelength scales, e.g. the plasma membrane: while it is lipid-dense and substantially different from aqueous cytosol, it is typically less than 10nm thick. One major exception to this is blood, and the hemoglobin of red blood cells in particular. Hemoglobin is strongly absorbing and scattering, and blood vessels much larger than capillaries (<100 microns in diameter) can cause quantitatively noticeable inhomogeneities.

Brain tissue is also very far from an ideal diffuser: the vast majority of photons are scattered only a few degrees, with the g parameter—the mean of the cosine of the scattered angle— often above 0.9^{15} . The effective isotropic scattering coefficient— $\mu'_s \stackrel{\text{def}}{=} \mu_s * (1 - g)$, qualitatively, the inverse of the length required to scatter enough to roughly isotropize the distribution—is thus on the order of 1 mm^{-1} . Heuristically, this is roughly the distance collimated light would have to travel to be scattered into a

nearly isotropic distribution. The typical scale of the brain structures, light sources, and penetration depths of interest to optogenetics are all submillimeter, so the ideal diffuser approximation is a bad one, and will result in erroneous values (i.e., too much angular dispersion) near the light source. I found this to be true in my Monte Carlo simulations, comparing runs using the effective isotropic scattering length and $g=0$ vs a realistic scattering length and g value (see below). This strong deviation from the assumptions in Aravansis is largely masked by virtue of the fact that they recorded only the total power and not its spatial distribution, and measured no closer than 0.2mm from the source.

Aravansis asserts that absorption can be ignored because scattering is much greater in brain tissue than absorption. This is something of a mistake. While the scattering coefficient is 10-100 times greater than the absorption coefficient, as noted above, the light is predominantly forward scattered. A more appropriate comparison is the effective isotropic scattering coefficient (also called the reduced scattering coefficient)—the inverse of the distance over which scattering would reduce the intensity to $1/e$ —and the absorption coefficient μ_a —the inverse of the distance over which absorption would reduce the intensity to $1/e$. The reduced scattering coefficient is smaller by roughly an order of magnitude in brain tissue at visible wavelengths, rendering the effects of absorption in a brain slice to be within an order of magnitude of the absorption, and not truly negligible.

Another troubling assumption about the tissue is the implicit assumption that the optical properties of the brain slice are quantitatively similar to in vivo tissue. This is a very difficult assumption to avoid, as effective in vivo measurements are difficult—see I.C for details. A major confound to this assumption is blood in the tissue: To keep the neurons—as well as glia and other cells, ideally—of the brain slice alive ex vivo, the brain is rapidly excised from the sacrificed animal and placed in an artificial cerebrospinal fluid (ACSF) solution that is bubbled with 95% oxygen/5% carbon dioxide. This does in fact keep neurons alive, even for many hours. It provides the necessary oxygen while maintaining an environment of the correct osmolarity and ion concentrations to keep individual neurons' physiology relatively normal. What happens to red blood cells (RBCs)—and the highly absorbing hemoglobin in them—is another question. Some of the blood will diffuse out of the brain slice and into the ACSF bath, but it is unlikely that all RBCs would escape the convoluted network of capillaries—the smallest of which literally stretch to fit even a single RBC—over even several hours. What fraction of the blood remains in the slice is an open question that Aravansis and others do not address. Indeed, Yaroslavsky et al.¹⁵ very responsibly make note of the fact that while they try to wash the blood out, their degree of success is uncertain.

The other major unknown about the blood is the degree of hemoglobin oxygenation. Deoxygenated hemoglobin's absorption coefficient is much lower across much of the visible range, especially towards the red. This uncertainty could be mitigated by measuring the dissolved oxygen content of the ACSF and using the well-known binding curve of oxygen to the hemoglobin tetramer. The oxygen saturation uncertainty is multiplied by the remaining RBC uncertainty, though, so such a calculation is not useful without also constraining the remaining RBC uncertainty. Neuron function hints at oxygen levels, as oxygen concentrations high enough to mostly oxygenate Hb are a necessary condition for healthy neuronal activity. Anecdotally, a typical slice in ACSF will have healthy neurons to a depth of perhaps 150 microns—which is a reasonable value to expect, as it is on the same order as the greatest nearest-capillary distance that is found in the brain¹⁶, so reasonable levels of oxygenation extend at least this deep. Measuring optical properties through tissue slices several hundred microns thick, however, will likely sample deoxygenated tissue, even if the ACSF is itself oxygenated. If the cells themselves are not

merely hypoxic but dying deep in these tissue slices, the changing morphology of the dying cells will also affect optical parameters as well. An extreme example of the effect of cell death on optical properties can be found in Yaroslavsky et al.¹⁵ where they report optical properties for both native (albeit in vitro) and heat-coagulated (80C for two hours) tissues.

The light source in any in vivo experiment also substantially deviates the planar, diffuse, and collimated assumptions: All light sources are finite in extent and will have regions near the light source edge where the K-M equation fails. For neuroscience experiments where the goal is to disturb as little tissue as possible, the (typically implanted) light sources are small, typically with emitting surfaces 50 to 200 microns in diameter, which is not approximately planar for essentially any relevant length scale.

Whether the light source is diffuse depends on the experimental setup, but in practice, our group finds the light intensity to be relatively uniform over the tip of fibers illuminated from lasers or LEDs. Of course, this is uniform only over the finite extent of the light source; as already mentioned, edge effects will not be accounted for in any K-M treatment.

Few light sources for in vivo experiments are collimated: Multimode optical fibers are commonly used, with numerical apertures (NA) typically between .2 and .4. In grey matter with a refractive index of ~ 1.37 ¹⁷, such fibers have acceptance angles of 8 and 17 degrees. These fibers are typically coupled to either lasers or LEDs of the appropriate wavelength. Typical commercial LEDs (whose emitting surfaces are intentionally roughened to increase surface area and/or to broaden the emission angles) have nearly Lambertian emission patterns, varying with the cosine of the angle with respect to the LED's surface normal. This dictates that the light entering—and exiting—the fiber will indeed be distributed up to the acceptance angle. Lasers can be much better collimated, but in practice our lab and anecdotal evidence from other labs observes that laser light coupled into a multimode optical fiber is also emitted across the full range of acceptance angles at the fiber tip for all experimental setups used. This can likely be understood as stemming from curving fibers that are imperfectly polished—all are hand-made for the particular experimental setup and polished by lab members. The lasers also have their coupling tuned to maximize the power exiting the fiber, without concern for collimation. The coupling optics are thus aligned to focus the laser beam down from its original beam diameter to a spot size less than or equal to the fiber core's diameter. This, by conservation of etendue, means the beam will have its angular distribution increased.

The Kubelka-Munk equation can be made to fit Aravansis' data well, but the scattering coefficient they derive is substantially different from the effective isotropic scattering coefficient found in Yaroslavsky et al.¹⁵, which is an order of magnitude shorter, at approximately 1.1 mm^{-1} . This might be due to the very different experimental contexts and mathematical frameworks—Yaroslavsky et al. used an inverse Monte Carlo method to fit their data, which involves fewer approximations.

To calculate the intensity along the fiber's axis, they added a geometric term due to the divergence of the light from the fiber's tip (radius r), based on its numerical aperture (NA) and the index of grey matter tissue n . This is a partial compensation for the deviation from the K-M assumption of planarity.

$$\frac{I_{\text{geometric}}(z)}{I(0)} = \frac{\rho^2}{(z + \rho)^2}$$

where

$$\rho = r \sqrt{\left(\frac{n}{NA}\right)^2 - 1}$$

This geometric correction assume the light is uniformly distributed across the interior of the light's cone—specifically, that it has the same value as if it were so distributed on the axis, where the equation is intended to be valid. As noted above, this is unlikely to be true, as the light is not typically uniform as a function of angle, nor is the path length to each point on a plane constant.

Yizhar et al.¹² reported a similar method for calculating light power as a function of distance from fiber emitter. In place of the photodetector, they placed a diffusing layer, which they then imaged from below, to get a two dimensional image of light distribution at a given depth. This allowed for off-axis estimates of expected light power, which deviate strongly from Aravansis' naïve conical-spread-moderated-by-scattering-losses model: as one might intuit, scattering causes relatively high fluence to be found outside the cone defined by the fiber's NA.

The Yizhar measurement system still had significant factors that can undermine its accuracy. All of the considerations about brain slice differing from in vivo conditions still apply. Additionally, any deviation from ideality of the diffuser would cause systematic measurement errors: if the likelihood of a photon to reach the imager is a function of incident angle on the diffuser, the measured fluence also becomes a function of photon direction.

The tissue also still has artificial boundary effects. In vivo, a cell can receive light that has passed it, but scatter back and reaches it from the opposite side from the excitation source (i.e., from a photon now traveling in the negative z direction). No single-sided photon detection device (such as a photodetector, objective lens, or diffuser sheet) could detect such a photon, as it would not impinge on the detector at all. This is one reason why advanced optical property calculation experiments use double integrating sphere configurations¹⁵: The sample is placed between two hollow integrating spheres that have two narrow holes on opposite sides to allow the beam to pass through them, with a final photodetector past the second integrating sphere. One sphere detects backscattered light, the other detects forward scattered light, and the additional photodetector detects ballistic photons still on the original beam path, thus covering nearly all 4π steradians and capturing nearly all photons. Figure 3 demonstrates the similar setup used by Yaroslavsky et al., where each component (ballistic, forward scattered, backward scattered) is measured in sequential experiments rather than in a combined double integrating sphere configuration.

The field of photodynamic therapy (PDT)¹⁸ has a similar interest in measuring fluence as a function of position in tissue: Calculating the rate of a photochemical reaction in a volume of tissue requires knowledge of the excitation fluence in that volume. PDT has been in use in its modern form since at least the 1980s, and the methods for measuring fluence in vivo are well-developed in that context¹⁹⁻²⁴, avoiding most of the problems seen in these attempts made in the context of optogenetics. A typical experiment uses a narrow probe with a small light collecting sphere that has near-isotropic sensitivity to measure the fluence at a given point^{19,22}. The experiments are performed in vivo and eliminate the uncertainties of relating excised tissue values to in vivo values, as well as eliminating artificial tissue

boundaries—especially important at the measuring boundary, which is the diffuser layer or photodetector in the experiments described above. The probes have nearly isotropic and homogeneous sensitivity to light hitting their outer surface, mitigating selective detection as a function of angle or position. Typical probes are optical fibers with detecting spheres on the implanted tip, coupled to photodetectors outside the tissue. The isotropically detecting spheres can be diffusers that scatter a fraction of the incident light down the fiber, where that fraction is nearly independent of the photon's original incident angle²². Another option is using a fluorescing sphere, where the light fluence to be measured is the excitation light and the fluorescence is detected by the photodetector after passing through a long pass filter that blocks the excitation light. As long as the fluorescence absorption and emission is isotropic—which is practically achievable—the measured fluorescence is proportional to the excitation light fluence passing through the sphere. In the case of fluorophores with long lifetimes, the system can be measured without a filter by using pulsed light and gating the photodetector to measure light power after the excitation pulse has completely passed.

Unfortunately, in the PDT context—the archetypical case being tumor treatment¹⁸, “small” probes have spheres 80 microns to 3mm in diameter. Not only does this limit resolution to substantially larger than a neuron and on the same scale as the light delivery device, the very presence of the probe displaces brain tissues enough to substantially change the tissue morphology—changing both the fluence distribution and the probe's position relative to a given piece of tissue. Additionally, the probe—both sphere and fiber—casts shadows and reflections within the tissue to be studied. Although it is a much smaller effect than with a planar detector (as in Aravansis and Yizhar), the size of the probe is, as noted, on the same scale as the variations we are measuring. In unpublished experiments, we obtained results of such light collecting sphere measurements that were unrealistic—likely due to these finite probe size problems, exacerbated by physical limitations of simultaneously implanting multiple fibers—for light delivery and detection—through skulls and into mouse brains *in vivo*. They were unrealistic in the sense that the measured fluence as a function of distance from the light source had unphysical forms: not monotonically decreasing, an almost perfectly linear decrease, rather than an approximately exponential plus $1/r^2$ decay, etc. In spite of the unphysical form of the results, the actual values obtained were plausible and agreed (within large experimental error bars) with my models. See I.C for these results, further discussion, and comparison with Monte Carlo predictions.

B. Monte Carlo simulations of light propagation and fluence *in vivo*

1. Motivations for selecting a Monte Carlo model

Although I began this work on determining light propagation for optogenetics before the above studies were published, I took a different approach in anticipation of the problems they went on to suffer from. In light of the limitations noted in I.A, I decided to work from experimental properties whose values were well-established and had convincing logic and method behind the experiments to measure them, namely: scattering coefficient, absorption coefficient, and phase function. These values are all a function of wavelength and tissue type. Another requirement of the model is that it make as few assumptions as possible that were likely to be violated *in vivo*. This meant, for example, that I eschewed Kubelka-Munk and similar models. Such models make assumptions in order to achieve a simple closed-form solution that are substantially at odds with the *in vivo* situations (see I.A).

These experimental properties do not allow the direct determination of the fluence as a function of position from a given light source. To do that, I needed to implement a model of the in vivo system that incorporated these properties to predict the result. This is certainly less satisfying than a compelling in vivo experiment, but as explained in I.A and demonstrated in I.C, such an experiment is difficult in a rodent brain. By starting only from well-measured properties and assumptions that are valid in vivo, I could largely confine the uncertainties in poorly known quantities to certain parameters of the model and clearly state what they were (see I.B.2). Additionally, I could make a range of predictions by running simulations of my model across the plausible range of the uncertain properties. An example is the uncertainty in blood fraction remaining in a slice of brain tissue used to calculate the optical properties: I ran simulations assuming no blood remained, and simulations assuming all blood remained and was fully oxygenated. These two extremes set realistic bounds on the actual in vivo situation. Setting such bounds is something much more difficult in the more direct experiments of Aravansis and Yizhar, where several uncertainties (blood fraction remaining in the tissue, impact of planar geometry, light escaping from the slice etc.) all enter the experimentally measured result in a way difficult to estimate.

For certain simple geometries—of both light source and tissue—closed form solutions or parametrized approximations could be found for a given set of optical parameters. However, in order to have the capacity to simulate realistic light sources and tissue geometries, I eschewed that approach from the beginning. Instead, I developed a Monte Carlo code to calculate light propagation from an arbitrary source into an arbitrary tissue volume. The Monte Carlo code stochastically samples from all possible photon trajectories, calculating the fluence passing through a given voxel of tissue. This is a commonly used method in many fields where an exact solution is not possible and a computation of the full state space (in this case, the full set of all possible photon trajectories) is intractable. In particular, it is frequently used to simulate light propagation, as is the case here.

2. Inputs and assumptions for the model

The Monte Carlo model I developed takes several inputs and makes certain assumptions. As noted in I.B.1, incorporating more elements of the model as inputs rather than assumptions allows for improved realism and characterization of uncertainty with this model.

One input is the distribution of photons from the light source in phase space—the position and momentum/direction of the photons as they enter the tissue to be measured. This is a function of the input light source and is controlled by the experimenter. It is implemented in the code as a user-defined probability function that the MC code samples from when generating initial random photon distribution.

Another input is the tissue geometry. This is a function of the region of brain targeted by experimenter, and is partially controlled by the experimenter, but also obviously a function of the native tissue's actual geometry and largely dictated by experimental necessity. The accuracy is limited to the accuracy of the geometry provided to the model. It is implemented in the code as an arbitrarily large rectangular prism of tissue divided into voxels, where the type of tissue present in each voxel is declared either algorithmically in the code or loaded from a data set (e.g., imported from a brain atlas). This explicitly incorporates multiple tissue types within the experimental volume. The simulated rectangular prism of tissue can be chosen to be large enough that boundary effects due to the simulation boundary are negligible. Alternatively, they can be intentionally utilized for some geometries. An example is simulating

a large planar geometry, where reflecting boundary conditions can allow a simulated volume with a small cross-section to accurately represent a large planar configuration.

A third input is tissue optical properties. This includes the absorption coefficient (μ_a), scattering coefficient (μ_s), and phase function for each tissue type present, at the wavelength of the light source to be simulated. The phase function is the probability distribution of scattering angles (θ) from a given scattering event. In biological tissue, one of a few forms of the phase function is chosen, and this function typically has a single free parameter, g , the anisotropy coefficient, which is $\langle \cos(\theta) \rangle$, where $\langle \rangle$ is the expectation value, an ensemble average over many scattering events.

The first assumption the model makes is obvious, yet not to be overlooked: it assumes that the inputs are accurate. See, for example, the discussion in I.B.3 of the uncertain impact of blood on the optical parameters of brain tissue. Another example would be speckle for laser excitation sources: I use simple parametrizations of the light entering the tissue that is reasonably accurate on average, but each laser and each fiber will have relatively static speckle patterns where the light intensity varies strongly as a function of position/angle. This speckle is typically rapidly smoothed by scattering in the tissue, but near to the fiber tip, the intensity variations can be large.

Another assumption is that the tissue volume modeled has a high enough resolution to avoid binning effects: The tissue voxels should be small compared to the size of the smallest tissue feature, the desired resolution of the fluence map, and the effective penetration depth of the light. Otherwise, tissue type boundaries that are curved or form planes at an angle to the voxel's Cartesian coordinate system will be "jagged" on a scale that will impact results. If the voxels are larger than the desired resolution or penetration depth, the resulting fluence map will have features of interest averaged out across the too-large voxel. The tradeoff is that smaller voxels require more compute time: In the limit of voxels much smaller than the scattering length, compute time increases as the inverse of the voxel length scale (see I.B.5).

Likewise, the tissue volume simulated must be large enough to avoid unwanted edge effects. A conservative estimate for the required size can be based on the absorption length. Absorption causes the fluence to decay exponentially with distance, so at a distance of a few absorption lengths, the magnitude of any fluence will be small and errors due to any chosen boundary conditions will likewise be small. Any scattering only serves to increase the effective path length needed to reach a certain distance, meaning basing the volume size on absorption lengths only will lead to a conservative estimate of what simulation volume is required. If multiple tissue types are present, a conservative estimate can be generated by using the longest absorption length when estimating the needed volume. Depending on the tissue types present this conservative estimate might lead to impractically large volumes, e.g., cerebrospinal fluid (CSF) will be much less scattering than gray or white matter and basing an estimate on it will lead to a large volume, even though the bulk of the brain is not CSF.

Any chosen volume (or voxel resolution) can be validated using the classic technique of (computational) parameter convergence studies²⁵: Computational parameters (such as simulated photon count, voxel size and total simulated volume) are at safely appropriate values if changing them does not alter the simulation's physical predictions. The case of voxel size is an example of a mesh convergence study. I do this by comparing two identical simulations that differ only in simulated volume (or voxel size, etc.): if the results show negligible differences in the volume of interest then the simulation has sufficient

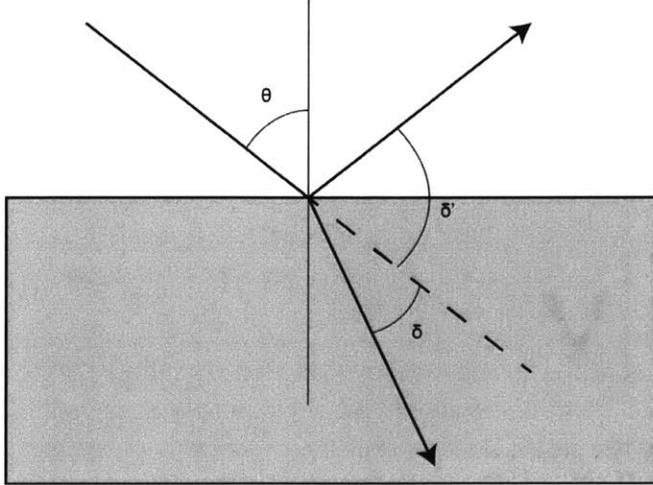


Figure 2 diagram of reflecting and refracting rays at a sharp tissue boundary

volume (or voxel resolution) to eliminate boundary (or binning) effects. While parameter convergence studies can confirm appropriate choice of computation parameters, given a specific set a physical parameters, they emphatically cannot confirm overall accuracy of the model.

My model does not incorporate refraction or reflection of light at boundaries of different tissue types with different refractive indices. The model thus implicitly assumes that these effects are quantitatively small. This is probably a good assumption: First, any reflections require relatively sharp transitions between refractive indices; any smooth

transition or interpenetration of tissue at the boundary acts as an impedance matching layer that reduces reflections. The impact of refraction and reflection can be quantified: The smallest index of refraction in the brain is found in the CSF. The CSF has a refractive index near that of water, $\sim 1.33^{17,26}$. The largest refractive index in the brain is found in tissue with high lipid density. The corpus callosum is typically the highest, as it is predominantly filled with fiber tracts of myelinated axons. It has a refractive index of $\sim 1.41^{17}$.

Take (for this analysis only) a planar boundary between these two materials with isotropic, unpolarized light incident on the boundary. If a photon is incident on this boundary with an angle of θ with respect to normal, refraction or reflection will cause the light to deviate from θ by an angle δ (refraction) or δ' (see fig.1). Snell's law can be used to calculate delta:

$$n_1 \sin \theta = n_2 \sin(\theta - \delta)$$

which rearranges to

$$\delta = \theta - \sin^{-1} \left(\frac{n_1}{n_2} \sin \theta \right)$$

While for reflection, the law of reflection leads to

$$\delta' = \pi - 2\theta$$

Fresnel equations then allow us to determine what fraction of photons are refracted or reflected for a given θ . Given that the brain is not a strong magnetic medium ($\mu_1 = \mu_2 = \mu_0$), we can calculate the reflectance for s- and p-polarizations as

$$R_s = \left| \frac{n_1 \cos \theta - n_2 \sqrt{1 - \left(\frac{n_1}{n_2} \sin \theta \right)^2}}{n_1 \cos \theta + n_2 \sqrt{1 - \left(\frac{n_1}{n_2} \sin \theta \right)^2}} \right|^2$$

$$R_p = \left| \frac{-n_2 \cos \theta + n_1 \sqrt{1 - \left(\frac{n_1}{n_2} \sin \theta\right)^2}}{n_2 \cos \theta + n_1 \sqrt{1 - \left(\frac{n_1}{n_2} \sin \theta\right)^2}} \right|^2$$

and the average reflectance (again, for unpolarized light), $R_m = (R_s + R_p)/2$.

We can now characterize the impact of the refraction/reflection on the photon's angle that is in some ways analogous to the characterization of the scattering phase function by the parameter g . Define f as the expectation value of the deflection: the mean over an ensemble of incident photons with direction chosen from an isotropic distribution (which is the uniform distribution over solid angle, but therefore not uniform as a function of θ). This is the average of the deflection, weighted between reflection and refraction by the reflectance:

$$f = \langle ((1 - R_m) \cos \delta + R_m \cos \delta') \rangle$$

The average is taken over incident angles from 0 to $\pi/2$. Isotropic light implies equal numbers of photons per element of solid angle $d\Omega = \sin(\theta) * d\theta * d\phi$. Therefore,

$$f = \int_0^{\pi/2} \left((1 - R_m(\theta)) \cos \left(\theta - \sin^{-1} \left(\frac{n_1}{n_2} \sin \theta \right) \right) + R_m(\theta) \cos(\pi - 2\theta) \right) \sin \theta d\theta$$

Where I have eliminated the integral over $d\phi$, which cancels with its probability normalization factor.

Using either $n_1=1.33$ and $n_2=1.41$ (low to high case) or $n_1=1.41$ and $n_2=1.33$ (high to low case), we get $f_{lh}=.99$ and $f_{hl}=.97$. f_{hl} deviates farther from 1 due to total internal reflection leading to relatively larger average deviations than the low-to-high case where reflection (and large refractive deflections) only become large at very oblique angles. These values of f are within a few percent of the value of g in various brain tissues. Indeed, it is larger than most tissues' values of g , meaning the average deflection due to refraction/reflection at a boundary is less than the average deflection due to a single scattering event. Over a single scattering length (often $< .05$ mm), tissue scattering will impact the photon's direction as much as crossing the tissue boundary.

There are two potentially important caveats to this analysis that reflection/refraction effects are minor: One is that the reflection/refraction effects are systematic: the distribution of light angles is changed in a coherent way: all photons are deflected towards (away) from normal as they enter the higher (lower) index tissue, and all reflection is back into the incident tissue. Scattering events deflect the photon in a random way. Thus boundary effects will take longer to smooth out than the values of f and g might indicate at first glance. Also, CSF is not nearly as scattering as most of the brain and demonstrates that a single scattering length can, in certain situations, be quite long.

The second caveat is that—while both are expectation values of comparable metrics—the expectation value is taken over probability distributions of different properties. In g , the probabilistic quantity is the scattering angle, a property of the scattering event. In f , the probabilistic quantities are the incident angle, a property of the photon population in the tissue, and whether the photon reflects. The “scattering” refraction/reflection angle is deterministic. This means that the value of f is dependent on the distribution of photons while g is not. Above, I have assumed the photons are isotropic—which, as

noted in several places in this thesis, is not true generally. If they have a different distribution, the value of f changes—and can decrease significantly in specific circumstances, such as nearly collimated light at very oblique incident angles, or (s-)polarized light incident near Brewster's angle.

Another assumption built into the model is that randomly selecting scattering angles from the correct phase function distribution of angles is an accurate way to model scattering in brain tissue. Notably, two photons with the same initial conditions will not follow the same trajectory in my simulations, as each scattering event is randomly chosen. Given that the scattering is dominated by Mie scattering that is refractive in origin, this is potentially not a safe assumption; if the tissue is static, two identical light waves will in reality propagate identically. However, movement at the microscale in living tissue sharply reduces the correlation time. Moving RBCs in particular disrupt the time coherence of the speckle pattern generated by a coherent light source. RBCs are highly scattering and absorbing due to their hemoglobin, and at the same time move relatively quickly. With a density of capillaries in the brain¹⁶ of $\sim 100 \text{ mm}^{-2}$, the average intercapillary distance is on the order of 100 microns. This is comparable to the scattering length, meaning that movement of RBCs will break time reversal symmetry over short (biologically speaking) timescales and sharply reduce correlation time over distances where scattering is appreciable. In addition, heartbeats (via both changes in overall intracranial pressure and local capillary diameter changes) and breathing lead to bulk and local motion of all neural tissue. This motion is large enough to substantially alter the microscopic optical paths and decorrelate photons that have otherwise identical initial conditions²⁷. Although the sharp reduction in correlation time due to RBC motion forms the basis for time-varying speckle imaging of blood flow in the brain²⁸, only changes (in measured flow rates and thus correlation times) are typically reported. Passing references to correlation times themselves indicate they are on the sub-millisecond²⁹ time scale in living brain tissue. This is as at or below the timescale of neural function (a spike generally taking approximately 1ms to for rise- and fall-times together; observed spike rates and local field potential oscillations are all less than—often much less than—300Hz, due to refractory periods and other biological limitations), and is at or below the timescale of typical optogenetic stimulation light pulses.

The model also assumes that the coarse-graining assumption of one set of isotropic optical parameters for a given tissue type is accurate, rather than needing to incorporate different optical parameters on a smaller, perhaps even micron-by-micron level within a given type of tissue. Bright field images of unaltered brain slices show that there is some variation between different areas of the same tissue type, and the tissue is not isotropic in laminar or striated tissue—which much of the brain is. The Monte Carlo model I developed can incorporate optical properties varying on small scale by simply shrinking the voxel size. The model can also be easily extended to incorporate anisotropies by making μ_s , μ_a , and g a function of direction. The difficulty would lie in determining the values of those properties over that scale. This assumption is also related to the one in the paragraph above: over very short length scales, a photon will encounter zero capillaries and can have longer correlation times. It will also encounter the same neural tissue and take the same optical path, though it might vary as a function of phase of heartbeat and breathing. Should this level of submicron detail be required, the Monte Carlo approach would break down. Models would need full wave simulations and extremely detailed anatomical data of the full volume. This is both impractical (at present) and unnecessary (at present) for most experimental scenarios.

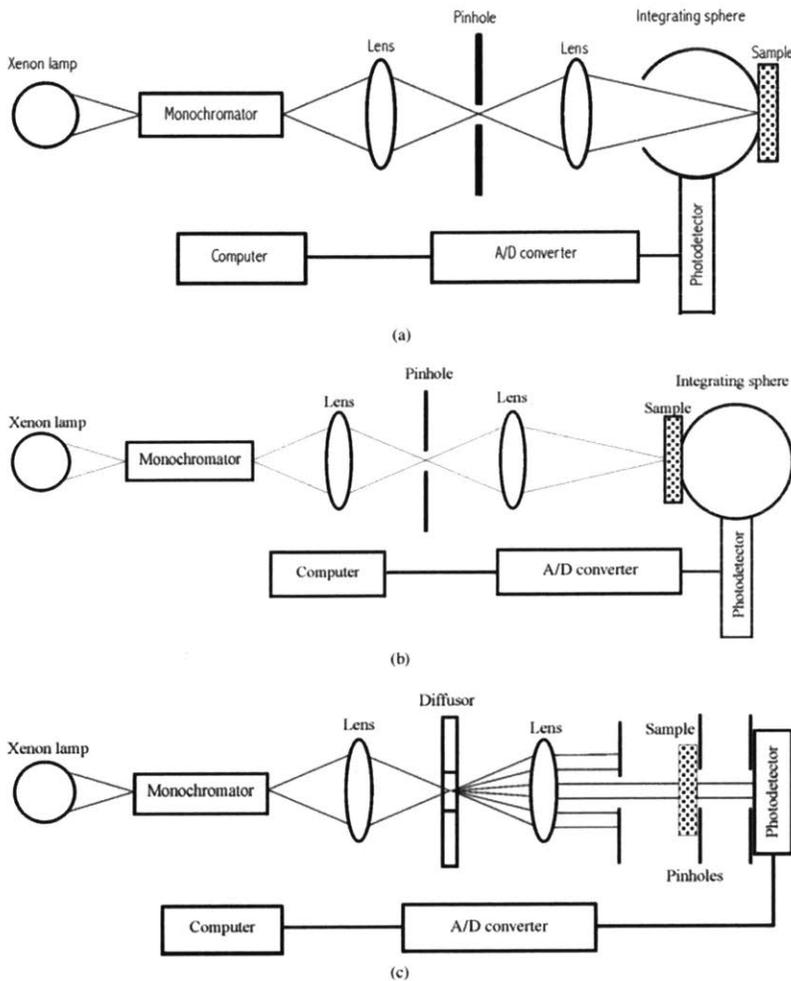


Figure 3 Diagram of experimental setup to measure scattering/absorption parameters and phase function, from Yaroslavsky *et al*.

As noted, laser sources are typically fiber coupled to get the light into the brain, and the emission from those fibers have strong speckle patterns. In principle, the speckle pattern can be simulated in my model with a detailed PDF of photon initial angular distributions, but this is impractical: the speckle pattern will change over time. Even temperature changes of the laser or optical parts coupling the light into the fiber will have a small effect, and any bumps of the fiber or motions of the animal—even when headfixed, a mouse or rat skull permits visible flexion, perhaps 50-100 μ m—will have larger impacts. Informal visual inspections show that any macroscopic motion of the fiber will shift the speckles by more than a speckle’s typical angular width, even though the overall pattern remains roughly invariant. We did not attempt

to include this in the model (or eliminate the speckles—by roughening fiber tips, for example) because the speckles are likely to be diffused over short distance scales by two mechanisms: The first and more important one is scattering in the brain. The myriad of subwavelength refractive scatterers are smaller than a typical speckle’s linear size, helping to diffuse the high intensity spot. The second mechanism is the changing environment over time: breathing, heartbeats, and any major muscle activity alter pressure in the brain, creating the motion artifacts that complicate many *in vivo* experiments. In this case, the motion of the scattering elements will cause the speckle to be scattered into different patterns at different time points. Breathing and heartbeats are relatively periodic, and the resulting variations in patterns won’t be ergodic, but they do strictly reduce the time averaged effect of speckle.

3. Choice of tissue optical parameters

The initial use of my Monte Carlo model in a publication was for a paper on systems for optogenetic manipulations³⁰. The goal was a quantitative but simple model of fluence as a function of position near an excitation LED, at the excitation wavelengths for the opsins ChR2 and Halo—taken to be 460nm and

590nm, respectively. For this purpose, I took the optical parameters reported in Yaroslavsky et al.¹⁵ for gray matter. See Table 1 for specific values used in this and other publications.

I selected the Yaroslavsky values because their experimental design conforms to best practices^{31,32} and their data analysis is careful and well-motivated^{14,32,33}. They performed three measurements on each sample, outlined in Figure 3. A first measurement used an integrating sphere with a hole for the beam on the input side of the sample to measure backscattered light. A second measurement with an integrating sphere on the output side measured forward scattered and ballistic light. A third measurement with a series of pinholes allowed only unscattered (ballistic) photons to reach a photodetector. These measured quantities were corrected for efficiency and angular sensitivity of the integrating spheres and photodetectors. To determine the tissue optical properties from the measured quantities, first a quasi-newtonian inverse model was used to get an approximate solution, followed by a Monte Carlo forward model to obtain more precise values. The Monte Carlo model could also account for light escaping, reflection at air/sample, sample/slide, and slide/air boundaries, etc. These models were based on an assumption that the Henyey-Greenstein phase function was accurate, which was justified by that phase function matching goniophotometric experimental results found in¹⁴ reasonably well.

To perform a simple validation of the Monte Carlo results in³⁰, we projected light from an LED like the one simulated through brain slices of varying thickness and imaged the light emerging. This technique is similar to the one later used by Yizhar et al.¹². We did not have a diffusing layer, though, so this check suffers from all the limitations of the Yizhar method noted in I.A, plus additional angle sensitivity in our measurements (any photons incident at too high an angle would not be captured by the objective lens with its finite numerical aperture). We found the contours at which the flux reached 10% and 1% of the initial value to be ~0.75 times the diameter predicted by the Monte Carlo model. The contours being smaller meant the measured fluence was less than predicted. This implied one (or both) of two things: The loss of photons due to the simplistic experimental design was having a quantitatively important effect, or the optical properties were incorrect. If the optical properties used in the simulation were incorrect, the fact that measured fluence was less than the predicted fluence implies some combination of μ_a is too low, μ_s is too low, and/or g is too high.

The parameters used for simulations in Chow et al.³⁴ were similar; they differed due to a careful re-interpolation of the data from figure 3A in¹⁵ to get the best accuracy possible. The parameters used for Kahn et al.³⁵ were identical to those used in Bernstein et al.³⁰. Despite the slightly different wavelength to be modeled (13nm shift), the re-interpolation from the Yaroslavsky¹⁵ plot appeared essentially the same. The difficulty of reading quantitative values from a plot—especially a (semi)log plot—makes a strong argument for publishing full tabular data with research papers whenever possible.

This Monte Carlo code was also used in Chuong et al.³⁶ This paper described the opsin known as Jaws, which was notable for being substantially red shifted from all other inhibitory opsins of comparable efficacy. As discussed further in I.D.4, this large red shift is important for two reasons. One is that availability of opsins with widely differing activation spectra is required for multicolor optogenetic control and/or readout: one type of cell expressing a conventional blue or yellow opsin and another cell type expressing Jaws would be independently addressable, or the same cell could express a blue light activated fluorescent reporter of activity while Jaws provided optogenetic stimulation. This concept has

since been realized in, e.g. Hochbaum et al.³⁷ A blue excitatory opsin expressed alongside Jaws would allow bidirectional control of a cell.

Chuong et al. focused on a second major benefit of a large red shift: beyond $\sim 450\text{nm}$, absorption (and to a lesser extent scattering) decreases strongly with increasing wavelength in most biological tissues. Several examples of this phenomenon can be found in¹⁵. An opsin capable of stimulation by readily available 633nm red lasers could be used with an entirely noninvasive stimulator, with light penetrating skin and even a thin bone such as a rodent skull as well as the brain tissue itself.

I was already concerned with how well the optical parameters measured in¹⁵ reflected the optical parameters of the same tissue in vivo, and this paper provided the impetus and opportunity for me to do a more thorough analysis. The primary concern was blood concentrations: Yaroslavsky et al. performed carefully designed experiments—but those experiments were in tissue slices, which they attempted to wash free of blood. Noting that the spectra bore some resemblance to a mixture of oxy- and deoxy-hemoglobin absorption spectra, they pointed out that, “although the samples were rinsed carefully in saline solution, it was obviously not possible to remove all blood residuals from the sections.” The quantity of blood present and its degree of oxygenation was uncertain. Given that a typical capillary diameter ($2.5\text{-}5\mu\text{m}$ ³⁸) is smaller than the diameter of an RBC, a simple wash is indeed unlikely to remove all the RBCs stuck in capillaries. A thorough transcordial perfusion with oxygenated, buffered saline to flush blood from the vasculature would be required to approach the true no-blood condition. This would need to be done before brain extraction, and would in turn raise concerns about brain slice health given the increased time between animal sacrifice and the brain slices being supported by well-oxygenated ACSF.

On the other hand, a solution with the same hemoglobin concentration as the in vivo brain ($\sim 1.8\%$ by volume, see below) would be expected to have a much greater absorption coefficient than is found by Yaroslavsky in their brain tissue slices. At $\sim 465\text{nm}$, Yaroslavsky measured gray matter to have $\mu_a = 0.07\text{ mm}^{-1}$ while dilute blood with 1.8% hematocrit would have³⁹ $\mu_a = 1.1\text{ mm}^{-1}$. At this wavelength oxy- and deoxy-hemoglobin have nearly the same absorption coefficient, so we can say that—barring some strong in vivo effect missed by various experimenters—a substantial amount of the blood has indeed been removed by the washing.

This could also explain the discrepancy between my simulations and the measurements noted above and described in Bernstein et al.³⁰: If Bernstein did not wash as much blood from the slice as did Yaroslavsky, the values I used for μ_a (and to a lesser extent, μ_s) would indeed be too low, fitting the hypothesis that erroneous optical parameters explain some or all of the discrepancy. Moreover, Bernstein was not attempting to wash out the blood—only follow a standard neuroscience brain slicing protocol focused on neuronal health, no absence of blood—so it is indeed likely that his slices had a greater concentration of RBCs.

To incorporate in vivo levels of blood into the optical parameters for Chuong et al.³⁶, I first asked, “what would the optical properties be of the blood alone?” RBCs are confined to within the vasculature, and that vasculature has a distribution of diameters spanning nearly three orders of magnitude. This would require extremely small voxels and thus large simulations to incorporate directly. Instead, I modeled the blood as a dilute liquid throughout the brain tissue, and thoroughly intermixed with the brain tissue, where both brain and blood caused scattering and absorption events independently. The quality of this

approximation of the blood as a diffuse liquid in the brain is not necessarily high: First, it assumes that coarse graining the vasculature does not introduce errors. This is likely not too bad as the mean intercapillary spacing is indeed much smaller than the absorption and effective isotropic scattering lengths. Secondly, it ignores the fact that at high blood concentrations—as in arterioles/venules and arteries/veins—the scattering coefficient saturates and becomes independent of concentration³⁹. That is, even if coarse graining is safe inasmuch as a photon experiences the average value of tissue+blood, the optical properties can still depend on how that blood is distributed. Additionally, any vessels with diameters that are a non-trivial fraction of an effective penetration depth will cast shadows and can only be treated by incorporating their presence into the voxel map of the tissue. However, my purpose was to develop a realistic-but-erring-on-the-side-of-pessimism scenario of how far light will penetrate, and this blood-in-brain-as-dilute-liquid approximation ignores any saturation of the scattering coefficient and thus can only underestimate the penetration distance.

The brain consists of about 4% blood by volume(BV)⁴⁰ and typical hematocrit (RBC volume fraction of the blood, abbreviated HCT) values are around 40-45%^{39,41}. 4% BV * 45% HCT means the brain is 1.8% RBCs by volume. Roggan et al.³⁹ demonstrate that—below 10% hematocrit levels—scattering and absorption are linear functions of concentration, and g is nearly constant. They also provide μ_s , μ_a , and g as a function of wavelength at 5% HCT. These data suffice to calculate the optical properties of my model of blood in the brain as a 1.8% HCT dilute solution present in the brain.

The next step was to combine the optical parameters due to blood and brain. The probability of a photon *not* scattering over a distance l is the probability of it not scattering off brain tissue and not scattering off blood. Assuming scattering by either blood or brain tissue to be independent, memoryless (Poisson) processes, this statement becomes

$$P(\text{no scatter}, l) = P(\text{no brain scatter}, l) * P(\text{no blood scatter}, l)$$

and the terms on the right have exponential probability density functions with parameters $\mu_{s,\text{brain}}$ and $\mu_{s,\text{blood}}$ respectively:

$$P(\text{no scatter}, l) = e^{-\mu_{s,\text{brain}}*l} * e^{-\mu_{s,\text{blood}}*l} = e^{-(\mu_{s,\text{brain}}+\mu_{s,\text{blood}})*l}$$

The effective scattering coefficient is thus the two component scattering coefficients added together, a result intimately related to the classic Beer-Lambert law for extinction coefficients. Determining the effective absorption coefficient proceeds identically, with the effective absorption coefficient being again the sum of individual absorption coefficients.

The calculation of g is only slightly more complicated: g is the expectation value of the cosine of the scattered angle; the scattering event can be caused by blood or by brain tissue:

$$g = \int_0^{2\pi} \int_0^\pi \{P(\text{brain scat.} | \text{scat.}) * f_{\text{brain}}(\theta) * \cos \theta + P(\text{blood scat.} | \text{scat.}) * f_{\text{blood}}(\theta) * \cos \theta\} \sin \theta d\theta d\varphi$$

The conditional probabilities are for the “next” single scattering event: What is the probability that the photon scattered once off brain tissue ($N_{\text{brain}}=1$) and zero times off blood ($N_{\text{blood}}=0$), given it has scattered once ($N=1$)? The conditional is fully dependent: if scattering from brain tissue has occurred,

scattering has occurred with probability 1, while scattering off brain and scattering off blood are assumed to be fully independent. The conditional probability becomes

$$P(\text{brain scatter}|\text{scatter}) = \frac{P(N_{\text{brain}} = 1) * P(N_{\text{blood}} = 0)}{P(N = 1)}$$

Inserting the appropriate Poisson distributions,

$$\begin{aligned} &= \frac{(\mu_{s,\text{brain}} * l * e^{-\mu_{s,\text{brain}} * l}) * e^{-\mu_{s,\text{blood}} * l}}{(\mu_{s,\text{brain}} + \mu_{s,\text{blood}}) * l * e^{-(\mu_{s,\text{brain}} + \mu_{s,\text{blood}}) * l}} \\ &= \frac{\mu_{s,\text{brain}}}{\mu_{s,\text{brain}} + \mu_{s,\text{blood}}} \end{aligned}$$

The argument is identical for $P(\text{blood scatter}|\text{scatter})$. The conditional probabilities are just the relative scattering coefficients. These are independent of θ and can be pulled out of the integral for g .

Separating the integral for each additive term yields

$$g = \frac{\mu_{s,\text{brain}}}{\mu_{s,\text{brain}} + \mu_{s,\text{blood}}} \int_0^\pi 2\pi * f_{\text{brain}}(\theta) \cos \theta \sin \theta d\theta + \frac{\mu_{s,\text{blood}}}{\mu_{s,\text{brain}} + \mu_{s,\text{blood}}} \int_0^\pi 2\pi * f_{\text{blood}}(\theta) \cos \theta \sin \theta d\theta$$

The integrals are now the values of g for the respective tissue, which are the experimentally determined values. g for the brain/dilute blood mixture system is thus the average of the two mixture components, weighted by relatively likelihood of scattering off each component:

$$g = \frac{\mu_{s,\text{brain}}}{\mu_{s,\text{brain}} + \mu_{s,\text{blood}}} g_{\text{brain}} + \frac{\mu_{s,\text{blood}}}{\mu_{s,\text{brain}} + \mu_{s,\text{blood}}} g_{\text{blood}}$$

g parametrizes the phase function f , but the phase function itself must be chosen. Following Yaroslavsky et al.¹⁵ 2002, I had chosen the Henyey-Greenstein phase function for all tissues to this point. However, Roggan et al.³⁹ used the Gegenbauer Kernel phase function when determining the optical properties of blood I used. In principle, it is possible to use separate phase functions for both categories of scattering events, but this is computationally very costly and adds code complexity. As elaborated in I.B.4, this would cost a performance penalty—in the best case—of a factor of two. Moreover, the Henyey-Greenstein phase function has been shown to be less accurate, at least in blood, than the Gegenbauer Kernel. Puzzlingly, this result was reported in Yaroslavsky et al. 1999¹⁴, which was the paper that group cited in Yaroslavsky et al. 2002¹⁵ to justify their use of the Henyey-Greenstein phase function in various tissues.

Given that the optical parameters for brain tissue I used were derived from experimental data under the explicit assumption of the Henyey-Greenstein phase function, using another phase function—even one demonstrated to be more physically realistic—is not necessarily safe. I evaluated the differences and demonstrated that the Gegenbauer Kernel as used by Roggan et al.³⁹ is the best choice, given the conditions.

The Gegenbauer Kernel is actually a family of phase functions with two parameters, g_{GK} and α :

$$f_{GK}(\theta) = \frac{\alpha g_{GK} (1 - g_{GK}^2)^{2\alpha}}{\pi((1 + g_{GK})^{2\alpha} - (1 - g_{GK})^{2\alpha})(1 + g_{GK}^2 - 2g_{GK} \cos \theta)^{1+\alpha}}$$

Roggan et al.³⁹ fixed the value of α to 1 and fit only g_{GK} to their data. Unless otherwise specified, the Gegenbauer Kernel referenced below has the $\alpha=1$ form. The Henyey-Greenstein phase function is a special case of the Gegenbauer Kernel, with $\alpha=1/2$. In this case (but not the general case) g_{GK} is in fact equal to g , the anisotropy factor. The phase function simplifies to

$$f_{HG}(\theta) = \frac{1 - g^2}{4\pi(1 + g^2 - 2g \cos \theta)^{3/2}}$$

To compare the two phase functions for a particular value of the anisotropy factor g , one must first determine the value of g_{GK} that yields the desired g . Both function can range from -1 to 1, so I generated a lookup table by computationally evaluating the integral

$$g = \int_0^\pi 2\pi * f_{GK}(\theta) \cos \theta \sin \theta d\theta$$

for 20,000 values of g_{GK} . Given that the resulting $g(g_{GK})$ function is smooth, slowly varying over increments of 1/10,000 and monotonic, this lookup table (and any interpolations) are more than adequate for finding an accurate f_{GK} for a given g . See I.B.4 for details.

A key wavelength to evaluate for the Jaws red opsin project was 633nm. It is within the Jaws excitation band while avoiding most other opsins/optical reporters's bands and can be produced using cheap, powerful HeNe lasers. Here, $g=.991$ ($g_{GK}=.9477$) for blood³⁹ and $g=.9$ ($g_{GK}=.781$) for gray matter¹⁵. These phase functions are plotted in Figure 4. While both of the phase functions have the same scattering anisotropy coefficient g , their angular distributions differ. Importantly, the deviation is much stronger at extremely high anisotropy coefficients. Therefore, using GK optical parameters with an HG phase function for blood would be less accurate than using HG parameters with a GK phase function for gray matter. I went on to use the GK phase function for all mixed blood/brain tissues.

Note also that while μ_s and g depend strongly on both the phase function chosen and the experimenter/experiment, the effective isotropic scattering coefficient $\mu'_s = \mu_s(1-g)$ remains quantitatively similar. This is because, for a given tissue type, experimental measurements that find higher μ_s also find higher values of g . Prosaically, whether a model says the photon scatters many times almost totally forwards or fewer times over a wider range of angles, the models rapidly converge on similar distributions of photon directions. Only at distances shorter than the longer of the two scattering lengths will the models' predictions be substantially different, and even the longest scattering length in the materials and wavelengths of interest (633nm light in gray matter) is only $\sim 110 \mu\text{m}$.

The process outlined here was performed for three wavelengths relevant to the paper: 532nm, 593, and 633nm. Existing inhibitory opsins had excitation spectra peaking in the green/yellow range, while Jaws worked effectively into the red. We then picked wavelengths in those ranges that had common and inexpensive light sources available (LEDs and/or lasers). Materials simulated were gray and white matter (with blood added)^{15,39} and bone^{42,43}. Only gray matter and bone at 532nm and 633nm were used in the paper as published. See Table 1 for optical properties as used in this and other papers described in this thesis.

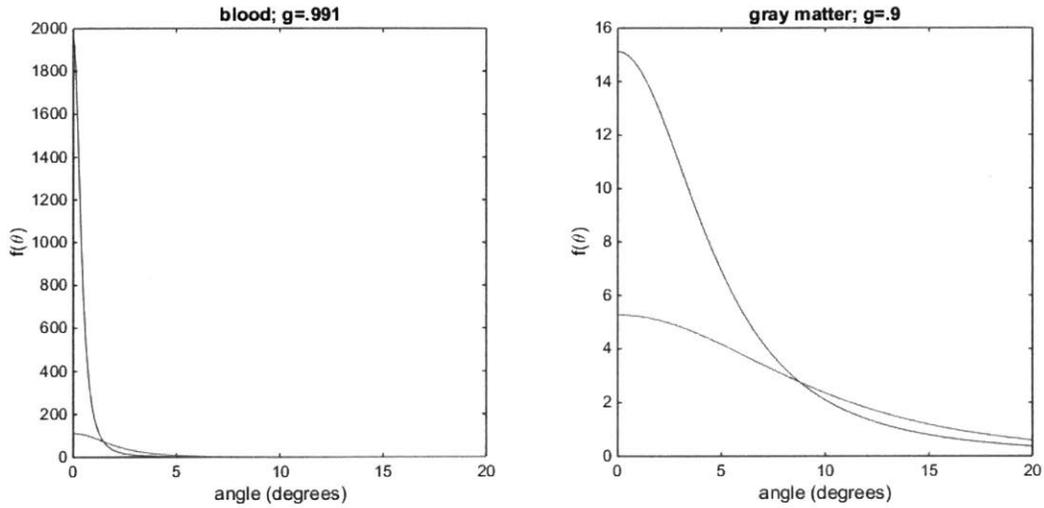


Figure 4 HG phase function, dark line; GK phase function, light line

Publication	Tissue	λ (nm)	μ_a (mm ⁻¹)	μ_s (mm ⁻¹)	g	Phase function
Bernstein et al. ³⁰	Gray matter	460	0.07	10	.88	Henyey-Greenstein
Bernstein et al. ³⁰	Gray matter	590	.027	9.0	.89	Henyey-Greenstein
Chow et al. ³⁴	Gray matter	593	.028	13	.89	Henyey-Greenstein
Kahn et al. ³⁵	Gray matter	473	.07	10	.88	Henyey-Greenstein
Chuong et al. ³⁶	Gray matter w/ blood	532	.942	23.3	.949	Gegenbauer Kernel
Chuong et al. ³⁶	Gray matter w/ blood	633	.071	20	.95	Gegenbauer Kernel
Chuong et al. ³⁶	Bone	532	.105	34	.93	Henyey Greenstein
Chuong et al. ³⁶	Bone	633	.06	33	.93	Henyey Greenstein
Chuong, not used	Gray matter w/ blood	593	.246	21.4	.948	Gegenbauer Kernel
Chuong, not used	Bone	593	.7	34	.93	Gegenbauer Kernel
Chuong, not used	White matter w/blood	532	1	55.3	.862	Gegenbauer Kernel
Chuong, not used	White matter w/blood	633	.134	53.8	.87	Gegenbauer Kernel

Table 1

4. Algorithm

The basic algorithm for the Monte Carlo light propagation is quite standard, based on⁴⁴ and similar to several GPU-enabled versions that emerged after I originally coded this version⁴⁵⁻⁴⁷. Scattering and absorption are treated differently for computational reasons, but the underlying physics is respected. The core loop proceeds as follows (see subsequent paragraphs for details), for all photon packets in parallel:

- 1) Advance the photon packet until the next event, which may be scattering, reaching a voxel boundary, or intersecting a boundary (such as the simulation boundary or a simulated measurement optical fiber).
- 2) Update the photon packet weight due to absorption
- 3) Record fluence contributed to each voxel from the photon packet

- 4) Terminate photons that have reached a boundary; perform Russian Roulette process on photon packets with very low weights
- 5) repeat until all photon packets terminate
- 6) save simulation parameters, 3D fluence map, and final photon packet locations to disk

For scattering, begin with a photon packet with a particular position and direction vector (this is a normalized velocity vector, as the actual photon speed is irrelevant for these simulations, and would only vary if the bulk index of refraction were included). Select a random distance to travel before scattering from the exponential distribution defined by the scattering coefficient. If that distance takes the photon into the next voxel, instead stop at the voxel boundary. Repeat this process until a stopping condition (photon has reached a boundary, or its weight is low and it is extinguished in the Russian Roulette process) is met. The photon is stopped at a voxel boundary because the next voxel might contain tissue with different optical properties. This throwing out of longer random distances is only permissible due to the memoryless nature of the exponential distribution. Were the probability distribution other than exponential, this biased sampling of the distribution would erroneously alter the distribution of distances before scattering. Calculating the odds of a photon propagating a distance at least d before scattering, where d is greater than the distance to the voxel boundary b but less than $2*b$: with no artificial voxels:

$$P(d) = \frac{1}{\mu_s} e^{-\mu_s * d}$$

with voxels it is the probability it has gone a distance b without scattering and an additional distance $d-b$ without scattering. Because those probabilities are independent (again, due to the memorylessness of the distribution):

$$P(d) = P(b) * P(d - b) = \frac{1}{\mu_s} e^{-\mu_s * b} * \frac{1}{\mu_s} e^{-\mu_s * (d-b)} = \frac{1}{\mu_s} e^{-\mu_s * d}$$

This generalizes in the obvious way to $3b > d > 2b$ and so on. Thus, the scattering distribution is unaffected by the introduction of voxels of any size. The memoryless property also conveniently allows the scattering algorithm to proceed identically regardless of what happened in its last step—whether it stopped at a voxel boundary or a scattering event, which is also helpful computationally (see I.B.5)

This photon advancing step does not advance all photon packets an equivalent amount of time. As time only enters into the simulation as an overall scaling by the photon emission rate^{*}, the simulation does

^{*} An aside on simulated time in the simulations: The transit time of a photon through a brain is orders of magnitude shorter than all (known to be) relevant timescales for optogenetics, and the light power densities are orders of magnitude lower than would generate noticeable nonlinear optical effects, thus rendering exactly when each photon was where irrelevant; time only enters into the simulation as an overall scaling factor of the rate of photon emission into the tissue. This text often refers to quantities having dimensions with inverse time but describes them with quantities lacking inverse time, such as power being the change in weight (equivalent to an energy) summed over all photons (dimensionless number). This is because the simulation does not specify what time duration it is simulating. The duration simulated (or equivalently, the emission rate) can be applied (if needed)

not need to do bookkeeping about position as a function of time. Indeed, this is what allows the massive parallelization that can be accelerated with GPU computing: The photons must all be doing the same thing (i.e., advancing until the next required scattering update), but they need not all be synchronized same simulated time. Time synchronization would require either advancing only the shortest amount of time that any photon traveled before scattering—the least common denominator among 1M+ simultaneously simulated photons, or simulating each photon independently and keeping track of position and velocity data at each step and interpolating to find the total fluence at a specific time. This would increase output size by roughly the average number of scattering events, on the order of 1000, and add a disk IO step to each photon packet advancement step. Disk IO is several orders of magnitude slower than the PCI-E transfers that the simulation needs as implemented, and would become a prohibitive bottleneck.

For absorption, each simulated photon packet is assigned a weight W , initially set to unity. Absorption decreases the weight, and represents the fraction of photons in the packet not yet absorbed. This is computationally very useful because it drastically reduces the number of photon packets required to get good statistics beyond an absorption length or two. The peril is “Monte Carlo Shot Noise”: if too few photon packets are simulated, the finite number will cause Shot Noise that is purely due to the small number of photon packets, not any physical effect. This noise can manifest at any distance, not just multiple absorption lengths from the source. As with many computational parameters, the number of photon packets required can be determined by increasing the parameter until results are quantitatively similar to the desired precision. Every time a photon is moved by a distance d (either to a voxel boundary or to a scattering event), its weight is reduced:

$$W \rightarrow W * e^{-\mu_a d}$$

If the photon packet’s weight is reduced to a very small value, it is desirable to terminate it and cease expending computational resources on it. However, in order to avoid distorting the simulation for this purely computational consideration to avoid affecting simulation results—in particular, to accurately map fluence far from the source, a Russian Roulette process is used to terminate photon packets: When a photon packet’s weight W falls below .0001, its weight is reset to 1 with probability W or terminated with probability $(1-W)$, where .0001 is chosen as a good balance between computational efficiency and avoiding detectable Russian Roulette shot noise, where a suddenly high weight photon leads to an unrealistically high fluence in a low-fluence voxel. In most applications I have used this code for including all those described in this work, more than 99% of simulated photons reach boundaries before triggering the Russian Roulette termination process.

To calculate the fluence F passing through each voxel, I first noted that the absorbed power density in a given voxel is simply the fluence multiplied by the absorption coefficient. The absorbed power is the change in weight of a photon packet in the voxel, summed over all simulated photon packets:

at all) to the simulation’s results. For example, to get an actual power density from these quantities reported in terms of energy density, one need only specify if the (say) 10M simulated photons were emitted over 1ms or 1ns, etc.

$$F * \mu_a = P = \sum_{i=1}^{N_{photons}} \Delta W_i = \sum_{i=1}^{N_{photons}} W_i (1 - e^{-\mu_a d_i})$$

$$F = 1/\mu_a \sum_{i=1}^{N_{photons}} W_i (1 - e^{-\mu_a d_i})$$

Where W_i is the weight at the start of each step, and d_i is the distance traveled in the voxel. d_i might be the distance to the voxel boundary or the next scattering event. If it scatters within the voxel, the next photon propagation step would add the additional energy deposited to that same voxel.

5. Implementation

Over several years, several versions of the code were used, as optimizations for various tasks and improvements in functionality were incorporated. I include a copy of the code as distributed in February 2016. All versions of the code employ the same core algorithms and the same workflow:

Inputs: A data structure containing computational parameters (how many photon packets to simulate, dimensions of each voxel in the simulated volume, size of simulated volume, etc.), a set of 3D arrays—one for each optical property (μ_s , μ_a , g , and phase function), and probability distributions for initial positions/directions of photons.

Propagation: The code copies relevant data to a GPU, which propagates each photon packet until it scatters or reaches a voxel boundary. It reduces the weight of each photon packet due to absorption. It keeps a running sum the fluence in each voxel as the photons propagate. When a photon stops (either by hitting a boundary or being removed due to low remaining weight), the code records its position, direction, and weight.

Outputs: A 3D array of fluence in each voxel—and because the fluence multiplied by the absorption coefficient is the power absorbed, this also provides the power dissipated in each voxel. A list of all photons final locations, directions, and weights.

I wrote the code for inputs, outputs, associated plotting tools, etc. in Matlab. I wrote the photon propagation code itself as a CUDA kernel to run on a GPU, taking advantage of the fact that each of millions of photons has the same set of mathematical operations performed on it—the work is “embarrassingly parallel”. The use of a custom kernel was required, as Matlab at the time (2008) had no support for the use of the GPU on builtin functions; it could only move data to/from the GPU and call custom kernels on the data it had sent to the GPU. Matlab’s GPU functionality has since expanded greatly: many of the relevant mathematical operations can be performed on the GPU, with Matlab’s inherent strengths in ease-of-use with matrices being very relevant to the task. Debugging the CUDA code and its interaction with Matlab was also extremely difficult at the time; Matlab did not report any debugging information from the CUDA process, nor was the CUDA debugger CUDA-GDB able to hook Matlab code. If the CUDA code experienced a runtime error, all the feedback provided was that it had failed to run. The use of CUDA was still a worthwhile decision, as the CUDA version ran 60-100 times faster than the same algorithm purely in Matlab. This code has run for thousands of wall hours over several years, making the initial difficult and time consuming debugging of the CUDA kernels well worth the cost. In the years after the initial kernel was built, only minor, quick additions of functionality

(related to boundary condition options) were ever needed, and they did not warrant a reanalysis of the design choices (e.g., whether it should be rewritten using the newly GPU-enabled Matlab builtins). Indeed, a custom kernel might still be the appropriate choice. The transforms on the data involve several steps and can involve several warp splits (see below) if not managed carefully, and any overhead from control moving repeatedly between Matlab program on the CPU and the math operations on the GPU is avoided with the CUDA kernel.

Were I to begin this project from scratch, I would elect to use an open source software stack, likely based around the SciPy (primarily, numpy/matplotlib) Python environment. The strong math support (BLAS, GPU accelerated functions, etc.), frequent updates (to, e.g., support better CUDA integration and fix bugs), and easier distribution (no end user needs a Matlab license) would likely outweigh the loss of some simplicity (e.g., Matlab's extremely simple and familiar plotting capabilities, its extremely quick code iteration time). I would continue to use CUDA even though it is not open source because it is both the most optimized and developer-friendly GPU acceleration API. OpenCL has become competitive in recent years and does not tie one to NVidia, but is still a little short of CUDA on those two fronts.

Two bugs within Matlab caused particularly pernicious problems that were hard to pin down and difficult to work around: certain builtin math operations unexpectedly fail on single precision data by outputting incorrect values. I used single precision in most places because typical GPUs can process them several times faster than double precision. Once the bug was determined, I worked around it by converting the data to double then back to single before transfer to GPU, resulting in a performance hit. However, being outside the primary photon propagation loop, the processing time penalty was less severe than the programmer time penalty incurred in finding the bug. Attempts to get Mathworks to patch the bug were initially unsuccessful, but they may well have by 2016. The most severe Matlab bug affecting this project is an error whereby Matlab reading an extremely large (>~ 2GB) file can result in corruption of that file. This occurs even when the file is set to read only in the filesystem. Matlab discussed this issue with me and determined that a single byte was being altered in the file, but failed to find a solution. The underlying bug may well lie in the operating system, but they failed to find any workaround. Being reliant on Matlab to make a fix was a very compromising position, especially in the face of OSS alternatives where I could have examined the relevant code and made my own patch to fix/avoid the problem. The workaround I use in the current circumstance is assiduously backing up these files and restoring any time corruption happens, which is roughly once every 100-200GB of files read.

The hardware used for this project was relatively basic: over the years I used three computers—as well as a home computer—to run the project. Each was a relatively high end desktop, but none had workstation- or enthusiast-class parts and avoided the concomitant major price increase. To run the CUDA code, an NVidia GPU is required, but consumer grade 3D video cards are sufficient and in most cases (notably excepting double precision calculations, which I did not require) rival the performance of the much more expensive Tesla and Quadro cards. The Monte Carlo code was even successfully run by collaborators on common laptops containing NVidia graphics accelerators. I used 1 or 2 TB spinning disk hard drives in mirrored RAID 1 configurations. RAID 1 not only protects the data from the failure of a single drive, it also reads data at twice the speed of a single disk by reading from both disks in the RAID array simultaneously. I typically used 32 GB of RAM, though this was for the computations in the ILM project (see II), rather than the Monte Carlo code described here. Video memory, on the other hand, was a limiting factor.

While the GPU could greatly outperform the CPU on the relevant floating point operations and on some L1 caching considerations, typical GPUs contained only 1 GB of RAM, as RAM is rarely a limiting factor in consumer 3D graphics rendering (for games, etc.). This limited the number of photon packets I could simulate before needing to transfer photon packet data between GPU memory and CPU memory via the PCI-E bus, which is far slower than on-CPU or on-GPU data transfers. I was able to find GPUs with 2GB of memory at reasonable prices and used those. Fortunately, GPU memory limitations only impacted speed; the overall problem size was not limited, because (in the low electric field linear regime) superposition allows calculation of each photon independently. For extremely large simulated volumes, the size of the optical property arrays could become problematic, but even then, one could configure Matlab to only pass the optical properties of the voxels containing a photon. This would once again change the GPU memory limit into a speed penalty (more CPU memory reads) rather than a problem size limit. The increased adoption of high resolution monitors, multimonitor setups, and VR has led to a rapid increase in GPU RAM in recent years; one can now buy a graphics card with 11GB RAM for less than \$1000; an instance of commercial concerns driving advances that scientists can and should take advantage of. This theme can also be seen in the prospects of directly implantable LEDs improving over time due to increased efficiency (see I.D.1) and the viability of ILMs as image sensor pixel pitch decreases (see II.B).

To use the code, one first runs `next_photon_input_struct.m`. This Matlab script simply assigns several input parameters to a single structure `s`. This structure should be saved alongside the output and can be considered output metadata. Knowing exactly which parameters were used to generate a given output is core to good data hygiene and reproducibility. Editing this script is the primary way to adjust computational parameters, and avoids needing to edit the main function to do so. This compartmentalization improves ease of use and reduces likelihood of introducing errors into the main function. Likewise, it avoids accidentally leaving out parameters from the metadata to be saved.

The second step is to run the function `volume_creator`. It takes as input the data structure `s` and the body of the function is used to generate the tissue volume to be simulated. This function's body can be programmed to read in a file (e.g., MRI data of a real brain) or otherwise generate the 3D arrays for each optical property. I frequently used it with simple Boolean masks for different tissue types to generate toy geometries, such as in³⁶ where I would use an expression to the effect of

```
is_skull=false(size(mu_a));
is_skull(1:round(skull_thick/dx),:,:)=true;
mu_a(is_skull)=mu_a_skull;
mu_s(is_skull)=mu_s_skull;
g(is_skull)=g_skull;
```

to set the optical properties of all voxels from the edge of the simulated volume to the thickness of the skull to take on the values for skull. `volume_creator` outputs a set of 3D arrays of optical properties for the volume to be simulated, and an enumeration of all tissue types and their optical properties. These data are automatically saved, as they are also important output metadata.

The main function, `nextphoton` may now be run. The first major computational task is calculating lookup tables for scattering angles. The phase function is a probability density function. To sample randomly from it, I employ a probability integral transform and apply it to a uniform (pseudo)random number

generated by the computer. The transform does not have a simple closed form and requires a computationally expensive numerical integration (to get the CDF from the PDF) to compute. A typical simulation of 10^6 photon packets will have on the order of 10^7 or 10^8 scattering events, each requiring a sample to be chosen from the phase function, so I create the lookup table for later fast access when calculating scattering events. Indeed, for certain applications, I have the code cache these lookup tables to disk, as hard drive reads are substantially faster than constructing the lookup tables even once.

Unlike much of the code, the generation of the CDF and creation of its inverse is done with double precision. This is because for large values of g , the CDF becomes nearly flat as the angle approaches π , and numerical noise became problematic with single precision. Even with double precision, occasionally numerical noise would lead to two computed values of the CDF at angles near π to be equal—or even decrease. While physically this is unsurprising—as modeled, the likelihood of direct backscattering becomes negligibly small and the CDF flat, modulo numerical noise—it renders the CDF uninvertible. To avoid this, I apply a smoothing filter to values of the CDF above 0.9—which improves accuracy because, as noted, the real CDF is nearly flat, smooth, and monotonic. Above 0.998 even smoothing results in occasional repeated values, so I simply do a linear interpolation between the points $[F^{-1}(.998), .998]$ and $[\pi, 1]$. $F^{-1}(.998) \approx .848\pi$ for all values of g in my work.

For the Gegenbauer Kernel, an additional step is required; the anisotropy coefficient g is reported in the literature, but the phase function is calculated using the parameter g_{GK} . As g is the expectation value of the cosine of the scattering angle, it also requires a numerical integration over the phase function (see I.B.3) and a calculation of the inverse. Once a table of $g(g_{GK})$ values is calculated, its inverse is used as a lookup table to determine the correct g_{GK} to use to calculate the CDF, as described in the previous paragraph. $g(g_{GK})$ is a smooth, monotonic function with domain and range of -1 to 1. For all values of g used, its derivative stayed within an order of magnitude of unity, and—unlike the CDF calculations—didn't have any numerical issues when inverted.

After some miscellaneous initialization and allocation of variables, the initial position and direction (i.e., its initial position in phase space) of each photon packet is assigned to a variable named `photonPhase`. This is done on the CPU as it—being a one-off operation—doesn't require intense optimization and is possibly not even faster with the overhead of loading a separate CUDA kernel to the GPU. It also makes good use of Matlab's internal strengths of matrix operations on large vectors: the phase space positions are selected from the desired random distribution, then translated and rotated to match the position and direction of the simulated light sources by multiplying with a 3D rotation matrix.

`photonPhase` is allocated as an N_{photons} by 8 array of singles, even though it contains only 7 values per photon ($X, Y, Z, V_x, V_y, V_z, \text{weight}$). This is done to optimize GPU performance, as the GPU memory reads perform much faster when the data is aligned to 128 byte L1 cache lines. Given that a single precision value takes 4 bytes, a read of 8 singles meets this requirement, while 6 out of every 8 reads of 7 singles will require reading two cache lines to get the necessary data. For $1/7 \approx 14\%$ more memory use, I use only $(8/(6*2+2)) = 57\%$ L1 cache bandwidth. As with many Monte Carlo problems, my code is memory bandwidth and compute throughput limited, making this a very good trade.

All of the necessary data is then copied to GPU memory. As noted, this is a performance bottleneck; a PCI-E 2.0 16x interface like the ones typically used between CPU and GPU has a theoretical throughput

of 8 GB/s⁴⁸, while a reference implementation of an NVidia GTX 560 similar to the ones I used has a theoretical on-board memory bandwidth of 128 GB/s⁴⁹. Not specified but often even more important, the latency (number of clock cycles between when data is requested by the processor and when the data arrives in the processor's registers) is at least an order of magnitude lower from GPU main memory than across the PCI-E bus, and perhaps two orders of magnitude lower from cache memory than across the bus. To minimize the number of times the PCI-E latency impacts performance and to "keep the pipe filled" with as much contiguous useful data as possible, I would launch as many photon packets at a time as would fit in GPU memory. This number was roughly $N_{\text{photons}} = 2 \cdot 10^6$, corresponding to each photon packet using somewhat less than 1KB RAM to store all of its data, which is about what is expected, given the data required for the computation. It is "somewhat less" than 1KB RAM because the shared data (of optical properties, for example) also resides in RAM, as does the memory for the display of the desktop; the same GPU running the Monte Carlo code was also acting as the display driver. I determined this number of photon packets by simple trial and error; if too many photons packets were sent, performance would drop precipitously as data was swapped constantly between CPU and GPU memory. The operation would often fail due to the GPU timing out waiting for data. As noted previously, if I wanted to simulate more than two million photon packets, I would run the code multiple times, summing the fluence and concatenating the final photon positions.

The Matlab code then loads the CUDA kernel itself and enters a loop that runs the kernel. Each iteration in the Matlab code calls the kernel once, and the kernel propagates each photon packet until the next stopping point (voxel boundary, scattering event, or termination event). Only a few arrays are passed between the GPU and CPU at each iteration. Matlab generates and passes to the GPU the set of random variables which will be needed in the next kernel call. At the time the code was written, this was relatively efficient on CPU rather than GPU because it would likely have required a separate CUDA kernel to be loaded and run before the Monte Carlo code. That situation is changing: the ability to easily and quickly generate of (pseudo)random numbers within the GPU is extremely valuable for many applications, including many of the machine learning applications that have proliferated on GPUs in recent years.

Most appealing for my purposes about generating the random numbers on the CPU is that knowing the values of the random variables substantially eases debugging. `photonPhase` is also passed back to the CPU after every kernel call. This also aids debugging and checking intermediate results, as well as allowing for checks on the validity of the current output (e.g. once every few million photon packets, single precision floating point error will lead to an attempt to take `acos(1.000001)` or similar and return elements of `photonPhase` as NaN). This could all be left on the GPU for a performance improvement. `photonPhase` is not passed back to the GPU, meaning this overhead is only one way. The final array passed to and from the GPU is `photonFlag`, which indicates whether each photon packet is still propagating or has terminated. A small percentage (<0.01%) of photons fail to terminate and, as debugging the CUDA kernel in Matlab is so obtuse, I simply end the loop when the number of photon packets still active is less than .01%. The uncertainty caused by doing this—even if the bias introduced is systematic—is much smaller than various other uncertainties in optical properties, etc. `photonFlag` is Boolean, so its size is only N_{photons} bits and takes a trivial amount of time to send across the PCI-E bus. However, the aforementioned latency still has full impact. A modest performance gain could be achieved by moving all the operations on `photonFlag` to the GPU.

CUDA is based on C and for straightforward, embarrassingly parallel problems such as this one, scarcely more complicated than regular C to use. One caveat to bridging the two programming languages, especially when referencing indexes of variables, is that Matlab uses 1-based indexing while CUDA, like C, uses 0-based indexing. While CUDA is simple to use, using the GPU efficiently requires some considerations not relevant to CPU programming. The code launches N_{photons} threads on the GPU, and the threads are grouped into “warps” (terminology borrowed from weaving) based on the number of CUDA cores on the device and the number of on-core memory registers required to run the desired operations. This means there is an incentive to minimize the number of variables used—and intermediate values required—in each thread, so that every core can be used with the given number of registers in each GPU processor. The number of CUDA cores was in the hundreds when this code was created and is in the thousands present, so a given kernel launch will launch hundreds or thousands of warps sequentially.

Each warp must also perform the same mathematical operations. This means that if a conditional (e.g. *if* or *while* statements) cause some threads of a warp to go down one branch while others go down another branch, the warp is split: while all threads going down branch A are computed, all threads going down branch B are halted until the kernel completes all threads on branch A. These warp splits are obviously very computationally expensive. Thus the use of conditionals—especially if threads are likely to go down more than one branch, and *especially* if it might result in several warp splits—is very expensive within the CUDA kernel. These performance penalties are so steep they can rapidly drop performance to below that of a single thread on the CPU (which both has higher single thread performance—even for floating point operations—and is optimized via more sophisticated branch prediction and the like to have vastly higher performance when it comes to conditionals).

Using extra floating point operations to avoid conditionals can be beneficial. For an example (that should actually be implemented with nested *fminf* functions),

```
float dshortest=dx*(dx<dy)*(dx<dz)+dy*(dy<dx)*(dy<dz)+dz*(dz<dx)*(dz<dy);
```

sets *dshortest* to be the minimum of *dx*, *dy*, and *dz* without any conditional statements. Note also that short-circuit multiplication decreases the cost of more complicated expressions by not evaluating any subexpression that has a prefactor of zero.

The CUDA kernel does contain a few *if* statements:

One *if* statement *returns* if the thread index exceeds the number of photon packets to be simulated. It branches in at most one (of the thousands) of warps—the last one. It does not result in a detectable performance hit and is only avoidable if the user specifies N_{photons} as a multiple of the number of threads in a warp, which is hardware specific and unreasonable to expect the user to determine.

Similarly, one *if* statement triggers the summation of all the fluence “deposited” by the photon packets. This occurs for a single thread at the end of the last warp as it is an atomic operation. It does not result in a detectable performance hit, and is hard to avoid. Performing the summation via atomic adds might also be possible, but might end up causing a greater performance hit due to the memory locks of the atomic operations on the actual data, rather than just on the increment variable.

One *if* statement is for a simulated light detector and stops a photon if it would hit the detector. How often this splits the warp depends on the size and location of the detector, but can occur at most once per simulated photon. Avoiding control structures here would require an extreme amount of extra floating point operations. Also, for most geometries I used (obviously including the most common case with no simulated light detectors), this rarely or never split warps.

One *if* statement branches based on whether the photon packet stops because it reaches a voxel boundary or a scattering event. The frequency of warp splitting here depends on the ratio of voxel dimension to scattering length, and will experience the most warp splitting when the two values are nearly equal. Here, too, avoiding control structures would require an extreme number of extra floating point operations. This is likely the biggest unavoidable warp split performance penalty.

The final *if* statement branches based on whether the photon packet has terminated or not. One branch is extremely fast as it does nothing if the photon packet has stopped, but it does still split the warp. This will split many warps when approximately half of the photon packets have stopped. I have experimented with ways to avoid this warp split, such as having Matlab submit only the lines of the photonPhase array for packets that have not stopped to the CUDA kernel. This prevents any warp splitting from this *if* statement, but resulted in a major performance slowdown. It appeared that Matlab would copy photonPhase to and from the GPU when doing this. In principle, this is not required (the full photonPhase is already present in the GPU memory), and perhaps with improved CUDA integration in Matlab this would be handled correctly today. This *if* constitutes a performance penalty that can be worked around should the need be great enough.

Another peculiarity of any multithreaded code such as CUDA is the indeterminate order in which data is processed. Thread 1 might process data element 1, but it might not do so before thread 327 processes data element 327. It might even do it simultaneously. For the calculation of fluence this is an important consideration. Two photon packets must not be allowed to update the value of the fluence in the same voxel at the same time; the result of one might overwrite the result of the other, rather than being additive—a problem of memory incoherency. The solution I chose was to have each thread report how much fluence it added, and to which voxel. Each thread also performed an atomic increment of a counter variable when it had updated its photon packet. `atomicInc` is guaranteed operate in each thread one at a time—and with a relatively small performance hit, considering it is an atomic operation. When the counter variable equals N_{photons} , it guarantees that all threads have completed updating all photon packets. The (one and only) thread that found this condition went ahead and added the fluence addition from each photon packet to the appropriate voxel. This is not parallelized so it runs no risk of memory losing coherency. It also performs only a small set of memory lookups and additions, so is not a major performance bottleneck; if it were, one could devise a chunking scheme where several threads each added fluence to only a particular subset of voxels.

Once the Matlab loop that calls the CUDA kernel reaches its stopping condition, the Matlab code pulls all outputs off the GPU, and the `nextphoton` function returns.

C. In vivo measurements of light fluence and heating

In order to validate the *in vivo* case and get experimental measurements of the effects of inhomogeneities in the brain, I and other group members attempted to use isotropic fluence probes *in*

vivo. As noted in I.B.3, this is the gold standard in the field of photomedicine/phototherapy, and we wished to apply the same here. The primary difficulty with this experiment is the rodent brain we wished to measure: it is small, soft (i.e., has low bulk and shear moduli), and located within a closed skull.

The probe must be as small as possible (to minimize mechanical displacement of the small rodent brain and to provide higher resolution measurements by detecting over as small a volume as possible) and as close to isotropic sensitivity as possible (to measure as close as possible to the true fluence, which will not be itself isotropic).

We attempted two types of device. One used a highly scattering sphere (specifically, plastic heavily doped with titanium dioxide) attached to a multimode optical fiber, where the fiber is inserted to nearly the sphere's center. The short scattering length rapidly randomizes the direction of the photons, rendering photons of all incident (solid) angles equally likely to enter the fiber. The only major anisotropy in this case is from the solid angle blocked by the fiber itself, which is minimized by using a small fiber. This probe is larger than desired because even this material has an effective isotropic scattering length of $>\sim 1\text{mm}$, necessitating a sphere of that radius or greater. 1mm being the average thickness of the entire mouse cortex, this was undesirably large. This design was discarded before reaching in vivo experiments.

The second type of device used a ruby sphere with a fiber attached to it. Rather than measure the light of interest directly, this device measured the ruby fluorescence induced by the incident light. To a good approximation, the incident light is absorbed isotropically by the ruby, and the fluorescence is emitted isotropically as well. Because this device does not rely on scattering to achieve near-isotropic sensitivity, it can be much smaller. We were limited by commercial availability to $\geq 0.2\text{mm}$ diameter ruby spheres. The attached fiber—being itself 62.5 μm in outer diameter—did block a larger solid angle (~ 0.29 steradians) on the ruby sphere than on the (larger) scattering sphere, but even so, only $0.29/(4\pi)=2\%$ of the incident solid angle was affected.

The fiber collects both fluorescent emission and the excitation light, most of which passes through the ruby sphere. While the excitation light is the light of interest, it is captured in a highly anisotropic manner by the fiber and confounds the fluorescent signal, which is the one that accurately reflects the excitation fluence passing through the ruby sphere. We considered three methods to separate the fluorescent emission from the excitation: One would be to simply filter the excitation light outside of the brain, between the fiber and the photodetector. Dichroic filters that can efficiently reject the excitation light are available, and the sources used (lasers or LEDs) have conveniently narrow band to make this straightforward. A second, similar, option is to detect the light with a spectrophotometer that can resolve the excitation and emission peaks. A third method utilizes ruby's unusually long fluorescence lifetime ($\sim 3\text{ms}$). This is much slower than the light source fall time (LEDs as used in our lab are often less than 10 microseconds, which is nowhere near any theoretical limit; that speed is just faster than timescales of neural firing and the kinetics of our optogenetic tools). There is some spread in the arrival time of excitation light due to varying path lengths taken by the photons, but with $n_{\text{brain}}\sim 1.4$ and μ_a in all cases greater than $.01\text{mm}^{-1}$, light traverses an absorption length in at most

$$t = \frac{n_{\text{brain}}}{\mu_a} \frac{1}{c} = 0.47\mu\text{s}$$

Any light arriving even a few microseconds after light source shutoff would have traveled many absorption lengths and be absorbed to negligible intensity. Thus, on the millisecond timescale of ruby fluorescence, the fall time of the excitation light detected by the photometer is essentially 0. Indeed, the measured signal was a large and rapid drop (90% intensity drop, <100 microsecond duration) followed by an exponential decay with $\tau \sim 3\text{ms}$. One limitation of the ruby fluorescence method not shared with the scattering sphere is that the light to be measured must excite detectable ruby fluorescence. Fortunately, ruby has two broad absorption peaks in the visible spectrum (peaks $\sim 420\text{nm}$ and $\sim 550\text{nm}$) that both have fast, non-radiative transitions to the fluorescing state. These broad peaks covered the desired wavelengths of the light to be measured.

I chose the last method because I did not have an appropriate dichroic filter or spectrophotometer to hand, and a few lines of code could easily extract the ruby fluorescence amplitude data from the recorded signal. This also allowed me to use the existing and very stable fiber-coupled setup on the photometer. A collaborator in the lab—Leah Acker—who ran similar experiments with an eye towards non-human primate applications worked in parallel on the spectrophotometric method⁵⁰.

I set out with an extremely meticulous approach to experimental design—measuring gray matter as far from any tissue boundary as possible, inserting the detecting ruby sphere with the attached fiber oriented to be as far from the excitation light source as possible, etc. This was, frankly, a mistake. While the goal was good, I set myself constraints that were nearly—if not totally—impossible to satisfy. In an animal with larger uninterrupted gray matter volumes, my design goals would have been appropriate, but for a basic model verification, anything larger than a rodent would have been practically and ethically hard to justify. Indeed, as noted, Dr. Acker was running fluence experiments in mice precisely to have the best data possible to extrapolate for primate use without needing to perform the experiment on a primate. Rats would have been an option for a rodent with a larger brain, but the cortical thickness of a rat is not substantially different than a mouse, both varying between $\sim .8$ and 1.2mm , depending on cortical region.

Dr. Acker's experiments progressed more quickly and to avoid duplication of effort, I and the rest of the group opted to use her experimental results. Her experimental geometry is shown in Figure 5: a 1.5mm diameter optical fiber—quite large in the context of mouse optogenetics, and thus identified as “Planar Illuminator; see I.D.4—was placed against the mouse brain after skull and dura removal. The fluence probe— $400\mu\text{m}$ multimode optical fiber with a $300\mu\text{m}$ ruby sphere detector on its end—was acutely inserted next to the planar illuminator, via the same craniotomy. It was inserted at a 23.5 degree angle to pass under the center of the “planar” illuminator, and she recorded several light measurements at intervals of 0.5mm from the plane of the illuminator. Each of these positions is noted with a red dot.

There are two examples of experimental difficulties—especially on the in vivo side—that demonstrate why it is important (even in computational work) to stay close to experiment. The planar illuminator requires a craniotomy with an area at least a few percent of the brain's surface. The fluence probe requires the craniotomy to be expanded further, and, as a practical matter, always with some exposed brain surface around it. In order to prevent cortical tissue from drying out during the experiment, artificial cerebrospinal fluid is dripped onto this craniotomy. This ACSF is not typically warmed above room temperature. Substantial evaporative cooling also occurs from this wet surface. Temperature measurements from the ultrasound project (**Error! Reference source not found.**) demonstrated that this

cooling can result in outer cortical layers falling all the way to within $\sim 2^{\circ}\text{C}$ of room temperature. The mice were also under general pentobarbital anesthesia throughout the measurements. Both cooling and anesthesia can drastically change blood flow and (via vasoconstriction/vasodilation) alter the blood fraction of the brain. These differences are likely not the largest sources of uncertainty in this project, and doing the experiment in vivo in the first place was a drastic improvement over the state of the literature, so we did not attempt to address them further. However, the insight about cooling from large craniotomies is generally overlooked by the neuroscience community. This oversight is quite troubling: a temperature change of even a few degrees can radically alter neuronal activity. Section II.C.5 addresses heating and cooling (and its aggravating factors of often-highly thermally conductive implants and sometimes active electronics on implants) is ancillary to the ILM project and its forthcoming publication, but actually represents an important contribution to the field.

Another major potential source of error is bleeding: damaging an arteriole while inserting the probe causes blood to wick around the entire probe, including the ruby sphere detector, and drastically affects results. This is not surprising, as concentrated blood is, as noted, extremely absorbing and scattering. On the optogenetics side, light delivery would be similarly impeded (and the impact of the optogenetic intervention reduced) by blood around the light delivery device. Dr. Acker was well aware of this issue, and between careful surgical technique and rejecting data from bloody probe insertions, ensured that bleeding did not corrupt her data. The experimental results are plotted in Figure 6. It was immediately clear that a naïve exponential falloff or “exponential falloff with inverse square” would not explain the in vivo data. Firstly, the data point at 0.5mm is near the edge of the illuminator, and any “planar” approximation is not good here. This is most clearly seen with the 532nm light having lower fluence at

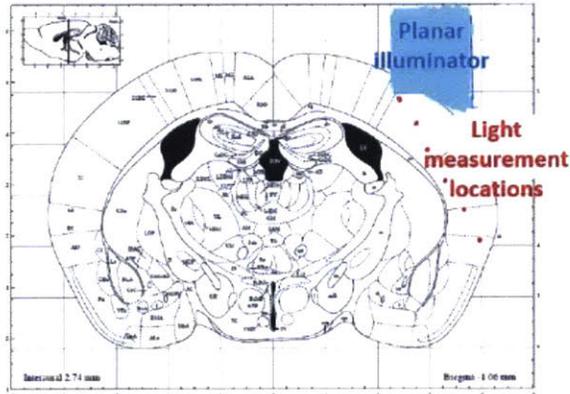


Figure 5 diagram of light measurement experimental geometry courtesy Leah Acker

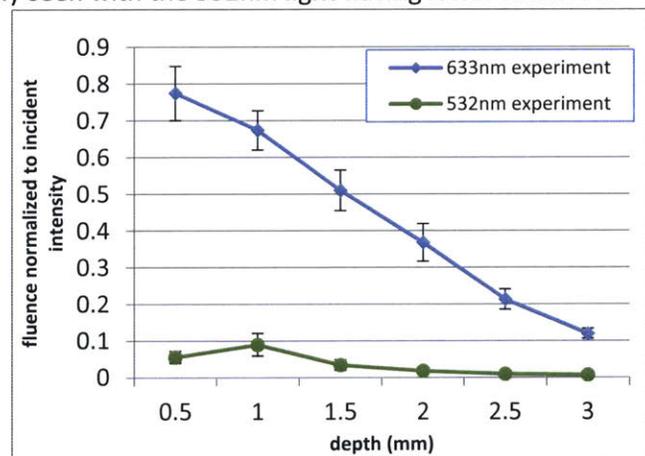


Figure 6 experimental results of light propagation in vivo courtesy Leah Acker

0.5mm than at 1.0mm: at 1.0mm, the probe is more directly under the probe’s illumination. This effect is less obvious in the 633nm light due to its much greater penetration depth with a roughly similar scattering length, but still present. This edge effect at shallow distances was of little concern for Dr. Acker’s work or the red opsin publication³⁶, which were both focused on long-distance light penetration where the probe was more centered, and the fluence varying more slowly in the transverse direction as well.

Beyond that geometric issue, the 633nm fluence as a function of depth is strikingly linear, while the 532nm light is—at 1mm and below—a good approximation of an exponential decay. Since the physical processes at play are the same for both wavelengths—just differences in magnitude of scattering and absorption, the difference should be strictly quantitative rather than also qualitative. Moreover, there is no reason to expect the fluence as a function of depth to behave linearly as it does for the red light. The “unexpected” linear behavior is actually not surprising in light of the experimental geometry of Figure 5: while all the tissue between the planar illuminator and the probe is gray matter, the probe’s recording sites pass close to the corpus callosum—a thick, dense white matter tract—and eventually close to the outside of the brain (where the boundary is successive layers of cerebrospinal fluid, dura, and skull). Both white matter and skull are highly scattering (white matter, for example has a scattering length less than half of gray matter, along with a smaller anisotropy factor—meaning each scattering event deflects the photon trajectory more), meaning more light will end up scattered back into the gray matter being probed.

The short absorption length of the 532nm green light means it doesn’t “see” much of these boundaries—most light is absorbed before reaching the boundaries, and more still is absorbed between scattering off the other tissue and reaching the probe—but the greater absorption length of the red light makes the boundary effects pronounced at depths >~1.5mm when the probe is in proximity to the corpus callosum or the skull.

These highly scattering boundaries can tell a consistent story of why the experimental red light data is roughly linear, but I simulated the system quantitatively to see if—and if so, by how much—my models were better than naïve techniques such as those used in Yizhar et al.¹². Dr. Acker also used simulations to derive the optical properties of the tissue from the model used in Yizhar et al.¹² to determine parameters with an approach independent of mine—albeit one with the incorrect geometrical assumption of a plane of tissue perpendicular to the light source. Dr. Acker’s best fit parameters are shown in Table 2, along with the parameters I used and the literature values I used to derive them.

633nm light parameters	Absorption parameter (mm ⁻¹)	Scattering parameter (mm ⁻¹)	Anisotropy coefficient
Dr. Acker’s in vivo parameters from experiment	.2721	2.126	.9
My in vivo parameters derived from literature ³⁶	.071	20	.95
Ex vivo blood parameters derived from literature ³⁹	.7	80	.97
Ex vivo gray matter (washed to attempt to remove blood) from literature ¹⁵	.017	9.2	.90

Table 2 estimations of optical parameters at 633nm

The parameters derived by Dr. Acker differed strongly from the ex vivo literature values. In particular, she found a scattering length five times longer than ex vivo gray matter, when one would expect the in

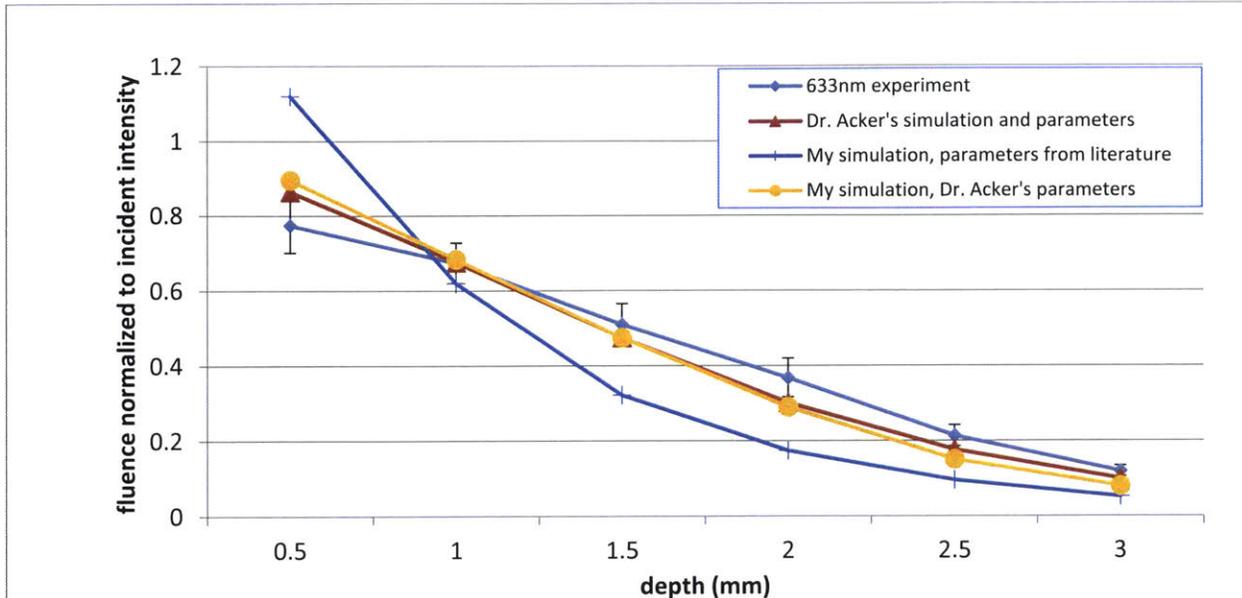


Figure 7 attempts to match simulation to experiment at 633nm

vivo case—having at least as much blood—would have shorter scattering lengths. That said, her model did provide a reasonable fit to the experimental data, while my simulation did not. Indeed, if I used my simulation code with her optical parameters, I achieved similar results. See Figure 7. Note, however, that although quantitatively good, the simulations were still qualitatively different from the experiment: the experimental data was very nearly flat, and all simulations—as well as physical arguments—suggest at least a roughly exponential decay.

Rather than believe that the outlandish optical parameters explained the data, I looked into incorporating the geometrical considerations mentioned above. While the tissue distortion caused by the craniotomy and probe insertion are difficult to quantify, my code could easily simulate a model of the skull and white matter and its impacts. Because the code sets the optical properties on a per-voxel basis, I set the relevant voxels to have optical properties of skull, gray matter, or white matter using a simplified geometry—a cross-section of which is shown in Figure 8 simulation geometry and experimental geometry overlaid on mouse brain atlas—and ran the simulation again. Also, the experimental data did not carefully document exactly how close to the “planar illuminator” the fluence probe was inserted, rendering the actual measurement position uncertain in the transverse direction. While of little importance at larger depths, this noticeably impacted the shallower measurements. Because my simulation calculates fluence in all voxels, variations in measurement position simply correspond to selecting different elements within a single simulation’s output data set.

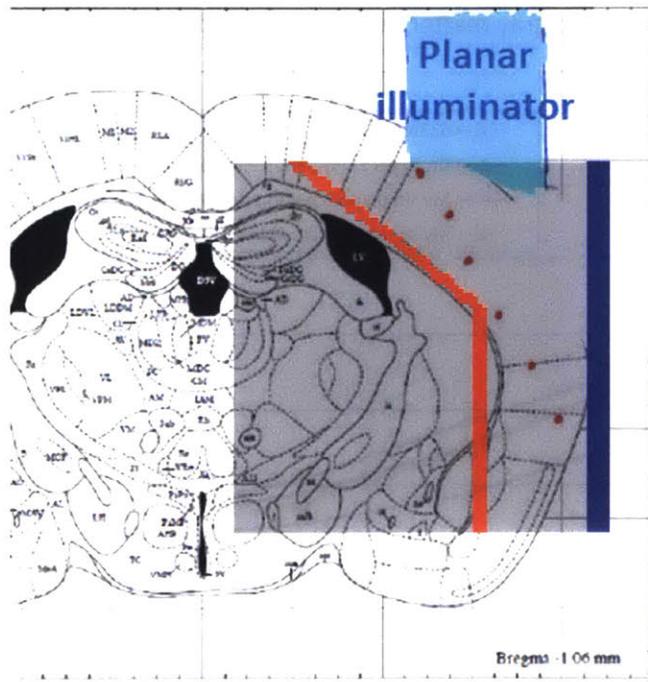


Figure 8 simulation geometry and experimental geometry overlaid on mouse brain atlas

Shallow depths are also disproportionately affected by anisotropies in the fluence detector sensitivity: The ~ 0.3 Steradians “blocked” by the fiber are centered only 23.5 degrees (the insertion angle) from the planar illuminator’s normal. I put blocked in quotes because the light is not actually blocked: all opaque covering layers are removed from the measurement fiber to reduce its diameter, leaving only the cladding and core. The light is confined in the fiber due to total internal refraction; both the cladding and core are transparent, and (at shallow depths) a nontrivial fraction of the excitation light reaching the ruby sphere is likely doing so by refracting through cladding and/or core to reach the ruby sphere. However, some sensitivity is lost to partial reflections at dielectric boundaries and refraction away from the detecting sphere. This effect is greater at shallow depths: the greater the

depth (and distance the excitation light has travelled), the more isotropic its distribution becomes due to scattering.

My code can also run simulations that partially account for this effect as well. By adding a geometry check to the CUDA code that detects when a photon packet intersects the detection sphere, those photons can be terminated, recording position and angle information of all such photons. That output data can then be used to cull photons that would not be detected by the real detection sphere. I implemented and tested a similar idea, but one that detected intersections with a flat disk—such as a polished fiber tip. Using just a fiber tip samples a much smaller region of phase space (collecting from only $\lesssim 0.5$ steradians, depending on fiber NA), but has advantages: It allows for a much smaller probe (only the $62.5 \mu\text{m}$ diameter fiber, rather than the fiber plus $200 \mu\text{m}$ ruby sphere glued to it). The probe is also simpler to build and use: the rubies needed to be attached with optical glue, with the distribution of glue further impacting the isotropicity of the fluence probe. The rubies were also prone to breaking off the fiber tip, ruining measurements—extracting the probe allowed too much bleeding and tissue displacement to allow quality measurements after reinsertion. Replacing the ruby also added a source of experimental variability between animals tested. Because we were attempting to validate simulations, we wanted to rely on extrapolations from those simulations as little as possible. Extrapolating from the small phase space sampled by a bare fiber tip relied more heavily on the accuracy of the models—and was more sensitive to noise in both experiment and in simulation (due to the finite number of photon

packets simulated), so we accepted the difficulties of the ruby sphere measurements and used that technique.

This geometry check slows the code by a noticeable amount—due both to the geometry calculations, and the additional warp splitting when some but not all photon packets intersect the sphere—but the reduced performance was on the order of a factor of 2, not orders of magnitude. This makes simulating a few key experimental geometries (not including per-voxel tissue optical parameters, which are already included) viable. However, by explicitly modeling the impact of the measuring probe in the simulation, that simulation is now only valid for that particular measuring probe in that particular position. As noted above, if the measuring probe is not included, calculating the expected fluence at various points corresponds to reading off different elements of the output volume, all from a single simulation. When including the measuring probe, each measurement probe position requires its own simulation. This introduction of additional free parameters to the simulation space (probe position and orientation) is the major increase in compute time for this technique.

D. Results

The Monte Carlo code I developed and the simulations based on it were used in a variety of projects. In some cases the simulations played an important part in published papers^{30,34–36}. In several other cases, the simulations were used by our group or collaborating groups in experimental design or project planning, such as^{51–53}. A typical example is shown in Figure 9, which was used to aid in determining appropriate waveguide apertures and pitch during the design phase. Some simulations of important cases were distilled into “rules of thumb” for back-of-the-envelope calculations and facilitated quick evaluation of early stage ideas. Additionally, the code provided all the simulation data to evaluate the implantable light field microimager (see chapter II). The earliest incarnation of my Monte Carlo code was actually made for a class project to simulate photoacoustic tomography data, for which it effectively calculated the excitation fluence that generated the ultrasonic response.

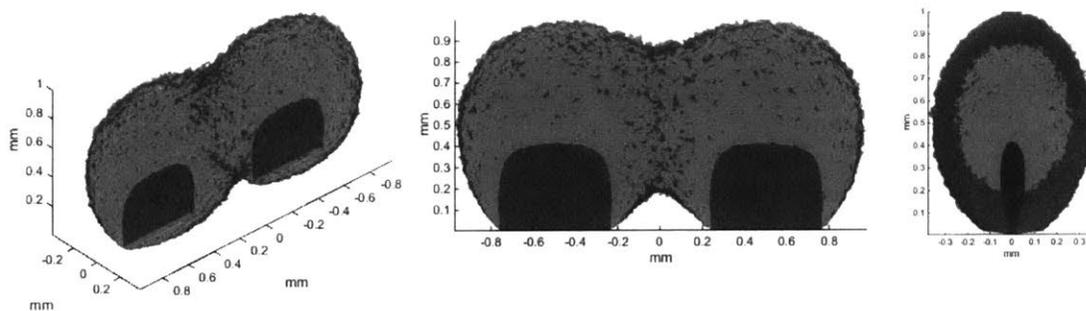


Figure 9 Sample views of a 3D render of two waveguide emission patterns. This simulation was provided to aid waveguide array design for Zorzos et al. 2010. Isosurfaces plotted at 10% and 1% of maximum fluence.

1. Contributions to “Prosthetic systems for therapeutic optical activation and silencing of genetically targeted neurons”

My simulations in Bernstein 2008³⁰ appear to be the first attempt to determine the in vivo fluence of light delivered for optogenetics. The fluence dose-current response curves of opsin ion channels are easy to determine in vitro—at least on a relative scale; determining the absolute current per channel is

much harder, requiring knowledge of the total number of active (correctly folded, inserted into the membrane, bound to all-*trans*-retinal, etc.) opsin molecules in the cell. Other papers had used electrophysiological measurements (along with the relative positions of excitation fiber and recording electrode) to determine what volume of tissue was affected at a given excitation laser/LED power. See, e.g.⁵⁴ These two sets of experimental data can be combined to yield an implied in vivo fluence distribution. However, there are important caveats to trusting this implied fluence: The electrophysiological activity is impacted not just by the value of the fluence illuminating the cell being recorded, but also by (spatially varying) expression levels of the opsin. Network effects due to the activity of neurons nearby also plays a major role, as clearly demonstrated in⁵⁴, especially figures 3 and S2. A spike in the recorded cell might be due to opsin activation in that cell, but it might instead be due to increased excitatory input from other optogenetically activated cells. Indeed, optogenetic stimulation of excitatory cells can lead to inhibition in that cell type, as optogenetically activated neurons drive inhibitory neurons that in turn synapse back onto the excitatory cells⁵⁴. This effect means that (in some cases) the opsin activation will be masked by inhibitory inputs and undetected by the in vivo electrode. The importance of independent estimates of fluence was recognized by other groups as well, leading to the later works noted in I.A.

The major result to come from Bernstein 2008 paper was estimates for the volume—and shape of the volume—excited by an LED. This was plotted as iso-fluence contours on a cross-section of the simulated volume passing through the center excitation LED. The contours were chosen to be on a log scale, normalized to the flux from the LED surface. This allows for intuitive mapping to a reader's desired experiment. For example, if one wanted to excite an opsin requiring $10\text{mW}/\text{mm}^2$ fluence at a distance of 1mm from an LED, the LED would need to generate $1\text{W}/\text{mm}^2$. The LED itself has a square emitting surface of side length l , so the single 2D slice doesn't fully capture the radiation pattern. The simulation itself included the square shape (and indeed the opaque wirebond pad for the anode that was located on the emitting surface as well), and we found other views (notably rotated 45 degrees around the emission axis, along the diagonal of the emitting surface, and perpendicular to the emission axis at varying distances) didn't add much information or change the qualitative story told by the views

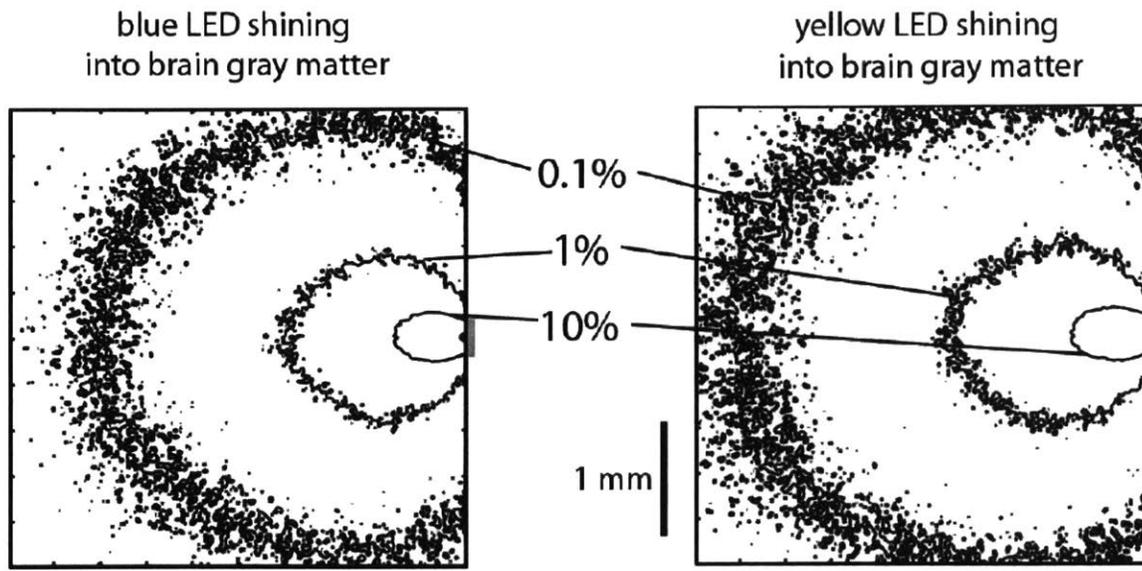


Figure 10 Normalized fluence contours from Bernstein 2008

presented: the only appreciable differences were 20-30 microns from the source (which, as one might expect, was on the same order as both $(\sqrt{2} - 1) * l$ —the difference between the width and the diagonal of the square LED—and the scattering length of the light).

The shape of the contours also gave rise to useful rules of thumb. The scattering renders the 1% and below contours to be nearly spherical. The 1% contour is ~ 1.5 mm diameter sphere centered 0.75mm from the LED, and—somewhat intuitively, given the importance of the exponential absorption—the 0.1% contour is ~ 3 mm centered at 1.5mm. Other simulations of optical fibers—with an angular distribution of photons limited by the fiber NA, instead of the fully Lambertian distribution of an LED—were nearly identical at those contours. Even changing the LED/fiber size (while keeping total light power fixed) had minimal impact on those contours, as long as the size remained less than the radius of the contour in question.

The “fuzziness” of the contour lines is due entirely to Monte Carlo shot noise. This could have been reduced by simulating more photon packets. It could also have been smoothed using any number of standard techniques (one straightforward option being a 3D convolution of the raw data with a simple, small kernel) without rendering the data misleading. We elected not to do either. The code was not yet well optimized with GPU acceleration, and—given the shot noise is reduced as square root number of simulated photon packets—reducing the fuzziness appreciably would have taken an excessive number of wall hours. Applying a simple smoothing filter to the data would have been computationally trivial, but we left the fuzziness as a subtle reminder that while the simulations accurately reflected the conditions of my model, there is uncertainty in how well those conditions (tissue parameter values, geometry of brain structures, homogeneity within a given tissue type, etc.) match a given *in vivo* scenario. Quantitative uncertainty measures (such as presenting models simulated with parameters at the top and bottom end of the uncertainty range of the literature range) would project a false sense of certainty about the uncertainties: As discussed in sections I.B.2 and I.B.3, the largest uncertainties are related to the difference between homogeneous *ex vivo* tissue and inhomogeneous *in vivo* tissue, not

the uncertainties in the ex vivo measured values. We explicitly did not assert an error range, plotted contours on a log scale—and mentioned our order-of-magnitude accuracy goal in the text, and allowed the fuzziness to remain in order to avoid (implicit) overclaiming. As noted in I.C, several attempts were made to validate or improve the parameter values—and quantify the inter-animal variability—but their success was limited. Additionally, the simulated volume of a 4mm cube of homogeneous gray matter is chosen for illustrative purposes, to display the intensity over three orders of magnitude and the geometry of the isosurfaces over one order of magnitude. No brain region in mice, rats, macaque monkeys, or other research animal has such a large volume of uniform tissue.

These simulations in combination with later thermal simulations ended up dooming the implanted LED approach described in the paper. The LED power required to illuminate a reasonable volume at a reasonable duty cycle was too high: the heat generated by the LEDs would lead to rapid, unacceptable temperature increases in the brain tissue near the implant. I consulted with Dr. Bernstein on the heating simulations he ran for this geometry (unpublished)—simulations that were essentially simpler versions of what I later did for the implantable light field imagers, as described in section **Error! Reference source not found.** He found that unless extreme active cooling—such as phase-change coolants—were applied directly to the back side of the implanted LEDs, the local brain temperature increase would be tens of degrees. Our group’s chosen target of at most one degree temperature change is somewhat uncertain—it is well-motivated by the literature (see, e.g.⁵⁵), but temperature effects on neural computation *in vivo* are systematically understudied—but these temperature changes were unambiguously unacceptable. Our group instead focused on implanted fiber arrays (as in^{51,56} *et al.*) or waveguide probes^{53,57}, both of which provide means of illuminating a large and/or addressable brain volume without placing the light sources—and their waste heat—inside the brain.

As an aside, the magnitude of LED efficiency improvements over the past several years indicate that this directly implanted LED idea could be revisited in the future. The nominal external quantum efficiency (EQE) in air of blue LEDs improved from ~.55 in 2008⁵⁸ to .85 in an LED made just 2 years later⁵⁹, and the light power density (at which the device can achieve the nominal EQE figure) has increased by a factor of approximately ten⁶⁰. The blue LEDs actually used in Bernstein 2008 had a nominal EQE of .4. The relevant figure of merit is the ratio of blue light power to heat power generated by the LED: how much light energy does the LED provide for a given amount of waste heat? Noting that essentially all of the power not emitted as light is eventually converted to heat within the LED structure, we can express this as

$$\frac{P_{light}}{P_{heat}} = \frac{\epsilon P_{total}}{(1 - \epsilon)P_{total}} = \frac{\epsilon}{1 - \epsilon}$$

where ϵ is the external quantum efficiency. The $1 - \epsilon$ in the denominator causes this ratio to increase without bound as the efficiency approaches 1. Indeed, using the nominal figures above, the ratio improved from .67 in our 2008 paper to 5.7 in 2010. For a given desired light intensity, marginal EQE gains near unity only marginally reduce the input power required to run the implant, but the waste heat is drastically reduced. Eventually, a different thermal limit is reached when the absorbed photons themselves generate enough heat to increase tissue temperature to the chosen threshold. This limit applies to any light delivery system. This limit is much less stringent, however, as the heat generated is distributed throughout a much larger volume (on the order of the optical penetration depth cubed,

rather than a thin contact layer surrounding the implant). The EQE figures cited are also for LEDs emitting into air; the situation in the brain is slightly better, as the brain tissue provides a better refractive index match than air, resulting in less internal reflection within the LED implant structure. Improvements to the EQE are uneven over time (and wavelength, with the “green valley” currently lagging both blue and red LED EQEs⁵⁸), so periodic surveys of the literature—and practical, commercially available LEDs—are warranted.

2. Contributions to “Characterization of the functional MRI response temporal linearity via optical control of neocortical pyramidal neurons”

Kahn et al.³⁵ used the same version of the code and the same optical parameters. The wavelength used was slightly longer than modeled in Bernstein et al. (473nm vs 460nm), but again, at the level of accuracy we desired and expected, the differences of optical properties was substantially smaller than other uncertainties. The difference here was that the source being simulated was a 200 um diameter, 0.48 NA optical fiber, rather than an LED. The 0.48 NA fiber emitting into $n_{\text{brain}}=1.37$ ¹⁷ cortical tissue confined the angular distribution to less than 20.5 degrees, but the light was assumed to be relatively collimated, and uniformly distributed over only 5 degrees. This was likely optimistic, given imperfections in optical setups used for neuroscience and the limitation imposed by conservation of Etendue (Liouville’s theorem, as applied to light): when the beam’s spatial extent is reduced to enter the small fiber, its angular extent must correspondingly increase, even if it was originally quite well collimated.

$$NA = n_{\text{brain}} \sin \theta_{\text{max}}$$

$$\theta_{\text{max}} = \text{asin} \left(\frac{NA}{n_{\text{brain}}} \right) = \text{asin} \left(\frac{0.48}{1.37} \right) = 20.5 \text{ degrees}$$

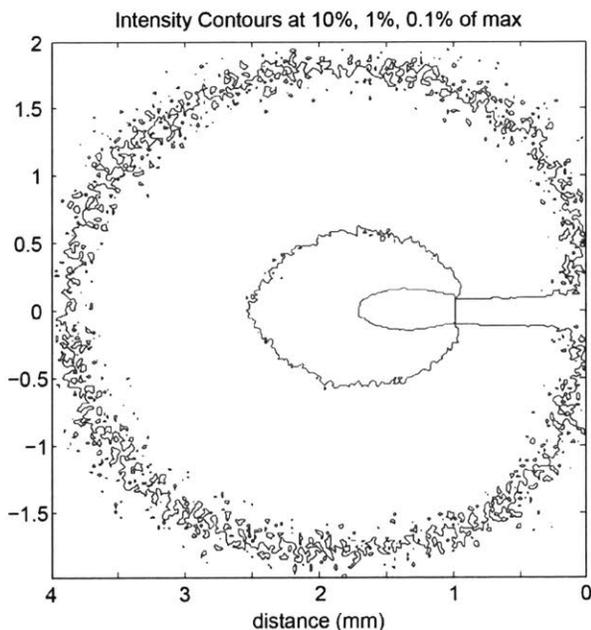


Figure 11 Fluence contours for nearly collimated light from a 200 micron fiber

Even if the laser beam was originally a perfect Gaussian beam (in fact any beam shape), imperfections in the polished fiber tips and bends in the fiber—which can vary from trial to trial or even moment to moment in *in vivo* experiments—will disrupt the original beam shape resulting in an interference pattern known as speckle. The altered direction and path length of different parts of the beam results in different optical path lengths to reach a given point, resulting in interference patterns that appear as speckles. These speckles are the variations of the angular distribution of the light intensity in both azimuthal and polar angles. As discussed in I.B.2, once it enters the brain, any beam structure is disrupted over a few scattering lengths in a time varying way. We do not believe it to be an important effect in a typical optogenetic experiment—though speckle patterns and their changes can be used to great effect when intentionally utilized, as in, e.g.^{27,28,61,62}.

The difference between the Lambertian distribution from the LED and the nearly-collimated distribution from the fiber is visibly discernable at the 10% level (compare the wider, shorter 10% contour in Figure 10 to the same contour in Figure 11), but barely perceptible at 1% and 0.1%. This is in keeping with the intuition provided by the effective isotropic scattering length—which is approximately 1mm: At shorter distances, the initial distribution of light appreciably influences the distribution in the brain, while at greater distances, the initial structure has been largely smoothed out and the light distribution rendered isotropic. Indeed, a rough rule of thumb used by our lab is that at fluence levels $< \sim 5\%$, the contours are spheres centered about one effective isotropic scattering length in front of the emitter.

3. Contributions to “High-performance genetically targetable optical neural silencing by light-driven proton pumps”

The discovery of the Arch protein’s power as an optogenetic tool was important for a number of reasons. Our lab demonstrated that it was an effective optogenetic proton pump: By pumping protons outward, it reduced intracellular voltage and inhibited spiking. We also showed—counter to what many expected for a proton transporter—that it did not disrupt the intracellular pH in neurons. It thus opened up a new, large class of potentially useful optogenetic molecules. However, this was more than a proof of principle for proton pumps; it was also a significant practical improvement over existing inhibitory opsins—which had been generally less powerful than their excitatory opsin counterparts. To illustrate this, our lab evaluated the silencing powers of Arch and another common inhibitory opsin, eNpHR, under a variety of conditions in cultured neurons. The cells were patch clamped and had a positive (excitatory) current injected, and the spiking rate was evaluated both with and without excitation light for the inhibitory opsin in question. A few data points of practical relevance were plotted (shown here in Figure 13) demonstrating at what irradiances and excitation currents the two molecules saw their silencing effectiveness decay. Note that in this culture experiment, the excitation light is well-collimated by the microscope and scattering through the single-cell thick sample can be neglected. In this regime, the fluence is equivalent to the irradiance.

We then set out to map these single-cell physiological results obtained in culture to practical implications for in vivo experiments. I ran a number of simulations to demonstrate what sort of shapes and volumes of brain tissue could be silenced at those landmark fluence values. I simulated a common optogenetic setup: 200um high-NA multimode fiber with an irradiance of $200\text{mW}/\text{mm}^2$ implanted into gray matter. I plotted both some standard values for simple scaling (e.g. $100\text{mW}/\text{mm}^2$ is the 50% contour) as well as the particular 1.3 and $0.35\text{mW}/\text{mm}^2$ fluences noted in Figure 13. This is shown in Figure 12.

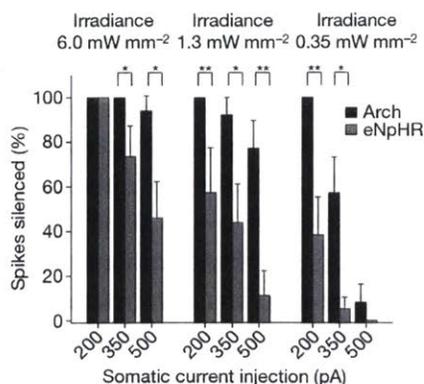


Figure 13 Silencing power of the proton pump Arch and a popular alternative, the inward chloride pump eNpHR, at three relevant irradiance levels, in cultured neurons. Taken from³⁴

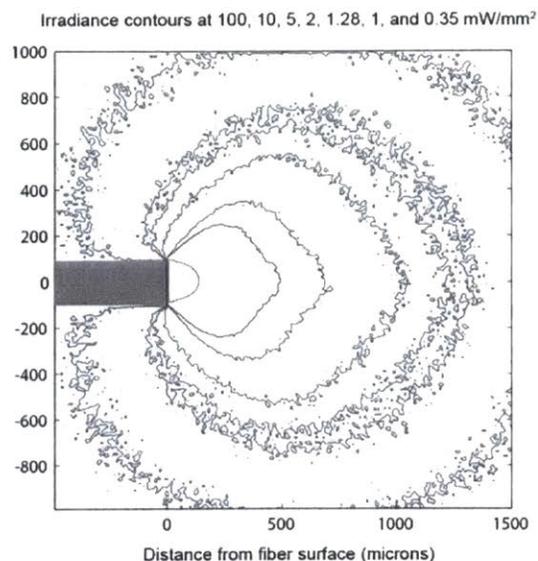


Figure 12 Simulated fluence contours at 593nm, a typical Arch excitation wavelength.

4. Contributions to “Noninvasive optical inhibition with a red-shifted microbial rhodopsin”

The proliferation of optogenetics as an indispensable tool for neuroscience led to a great demand for opsins with different excitation wavelengths. On the one hand, there was a demand for independent addressability: target two different neuron types with opsins using different colors to control them both independently⁶³; put one inhibitory and one excitatory opsin in the same neuron type to gain full, bidirectional control over its spiking/intracellular potential⁶⁴; put an opsin and a reporter (a fluorescent voltage or calcium sensor) in the same cell^{37,65}, exciting the reporter nearly continuously for readout while exciting the opsin with the time course defined the experimental protocol; and so on. On the other hand, there are very important physical considerations regarding excitation spectra: these include factors such as two photon cross section (and its overlap with commonly available/affordable femtosecond pulse laser wavelengths), but possibly the single biggest is how the excitation wavelength propagates in the brain. As easily seen in¹⁵, the scattering properties of gray matter washed of blood change by a factor of a few and absorption by an order of magnitude across the visible spectrum. And—as discussed at greater length in I.B.3—blood is even more sensitive to wavelength³⁹, and plays a key role in the highly vascularized tissue of the brain.

The goal of our paper here was to enable an entirely new category of optogenetic experiment: effectively noninvasive optogenetic stimulation, without any excitation light delivery mechanism in contact with the brain. In the case of transgenic mice, the mouse would never have surgery that penetrated the skull or brain, and even in the case of electroporated/virally infected opsin delivery, the surgery (involving a needle smaller than a typical implanted optical fiber) would be weeks before the experiment and could completely heal, leaving a fully intact mouse for the experiment.

To achieve this, we would leverage the somewhat longer scattering lengths and much longer absorption lengths of gray matter at wavelengths longer than 600nm. By focusing on proteins derived and optimized from natural sources, we were limiting our available spectrum to a range extending only

slightly beyond visible, because the vast majority of naturally occurring light sensitive ion channels and pumps are sensitive to that range. It is possible to shift the excitation spectrum using protein engineering techniques such as directed evolution and rational design, but these methods tend to yield small shifts, especially when also requiring useful photocurrents. Other techniques—possibly de novo protein design, discovery of exotic opsins, and (perhaps more likely, and the subject of intense research) non-protein techniques like membrane-targeted quantum dots/nanostructures—will likely enable a wider range of possible excitation spectra in the future. That said, this paper demonstrated the existing “find and optimize naturally occurring opsins” paradigm can be used to develop opsins that work in the non-invasive experimental regime while retaining all the benefits of using the genetic engineering toolbox. This does require the aggressive, broad search for useful opsins (as, for example, with the discovery of Arch³⁴ and the viability of proton pumps), combined with genetic engineering techniques (which generally have more headroom for improving photocurrent, temporal properties, and concerns like membrane trafficking than they do for spectrum shifting).

As with other experiments, I provided the calculated fluence data that corroborated physiological measurements and extended them to regimes hard or impossible to measure with direct experiment. This required the more nuanced treatment of in vivo brain parameters, already detailed in I.B.3. I also incorporated skull for the first time, based on experimentally determined values reported in^{42,43}. Bone has a somewhat higher scattering coefficient, but comparable or lower absorption coefficient to gray matter, as might be expected for a white, less-vascularized tissue.

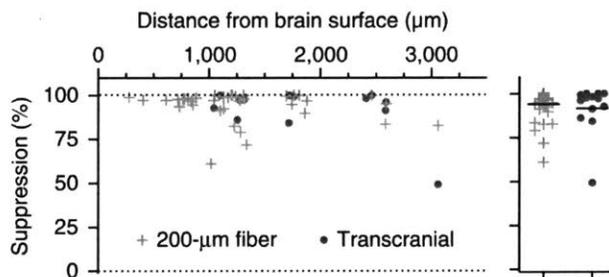


Figure 14 In vivo patch clamp measurements of spike inhibition. Data from noninvasive light source and fiber implanted near the target neuron both shown for comparison. Taken from³⁶

I did not simulate skin or hair, and the skull was simulated to have been thinned to 200 microns. This reflects the experimental reality that the external excitation source would have to be held in place. A typical way to do this is to resect some skin, thin the skull, and glue (with dental acrylic and/or optical adhesives) a canula that can hold the excitation fiber to the skull. Skin’s effect would be on the same order as gray matter, while mouse hair is extremely highly scattering and (at least for dark coats) absorbing

and would likely need to be shaved regardless of other experimental requirements.

Figure 2Figure 14 demonstrates the power of the Jaws to suppress neuronal spiking in vivo, both transcranially and with a conventional inserted optical fiber for comparison. Figure 15 shows my

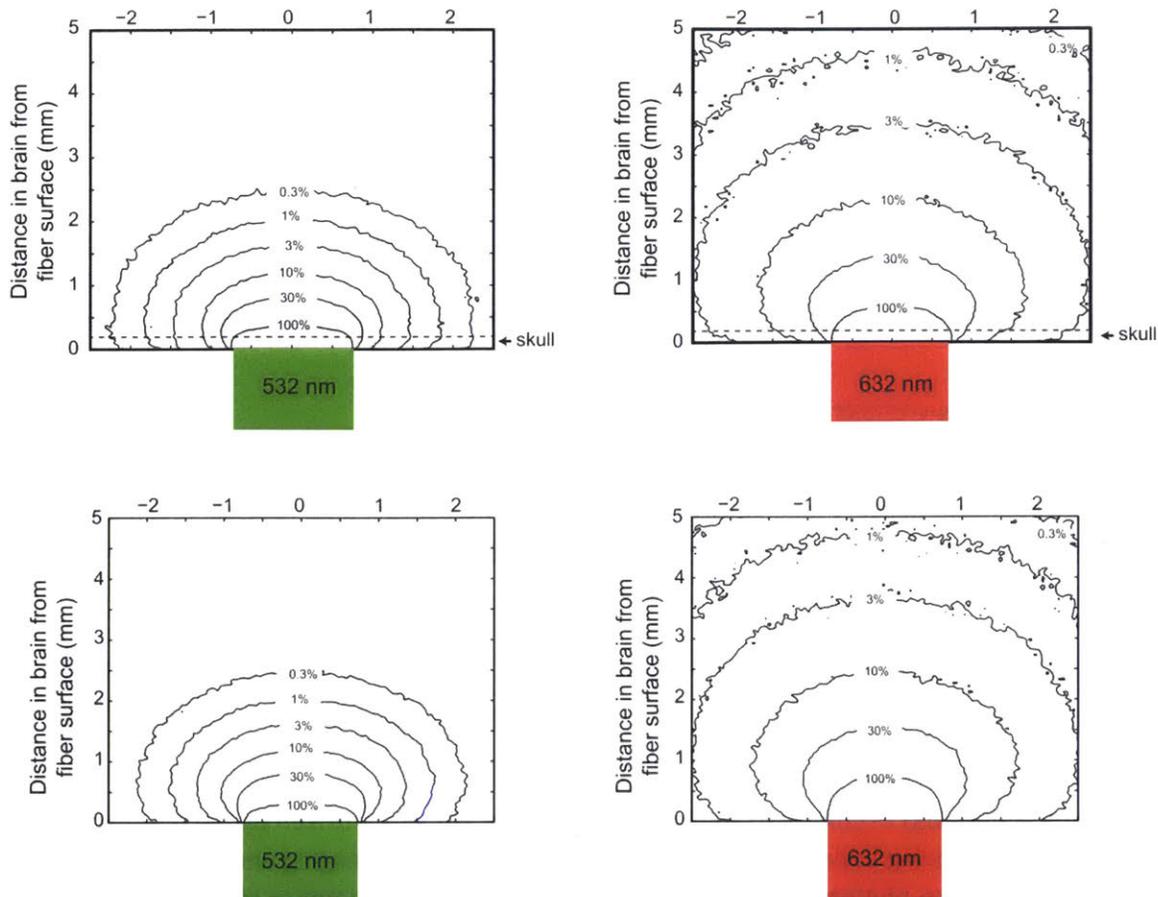


Figure 15 Fluence contours of simulated green and red excitation light, quantifying the benefits of moving to the red end of the spectrum. Simulated excitation light delivered at the thinned skull (top) and at the dura (bottom)

simulations quantifying how well red light penetrates in comparison to a green wavelength used by other inhibitory opsins. That figure makes apparent why noninvasive opsin excitation was rarely attempted before this publication. At high enough light powers to reach sufficient fluence at deep targets, the brain heating due to the absorbed light becomes unacceptable. Figure 16 shows experimental results of light measurement. They are roughly in agreement with simulation. The methodology—and caveats for experimental fluence measurements can be found in I.C.

We simulated—and used in experiments—1.5mm diameter plastic fibers, very large in the context of mouse optogenetics. This was to have them be something approximating a planar illuminator. This was to improve depth of stimulation by minimizing the roughly inverse square falloff of small sources. That is one tradeoff of the noninvasive method presented: At these distances of multiple effective scattering lengths, especially through skull, nearly all spatial structure is lost. One can still achieve spatial localization by choice of viral injection site when initially delivering the opsin, and in some cases by confining expression to the desired areas. Implantable fibers and fiber arrays do add a powerful ability



Figure 16 in vivo fluence measurements of 532nm and 632nm light

to provide better spatial addressing. This loss of spatial resolution caveat also does not apply to some more sophisticated light delivery mechanisms: multiphoton microscopy, time reversal techniques, adaptive optics, and other techniques for fluorescent imaging through turbid media can all potentially be used for exciting opsins instead of fluorophores.

II. Implantable light field microimager: designing and evaluating a new brain imager

A. Introduction

Imaging neural activity has become extremely important in recent years, driven by the production of novel and rapidly improving fluorescent indicators of neural activity⁶⁶. Imaging neural activity at depth, however, is difficult. Two-photon microscopy can image down to a millimeter or so⁶⁷⁻⁶⁹. This reach can be extended by using endoscopic devices that image at the ends of implanted GRIN lenses, fiber bundles, multimode fibers, or prisms⁷⁰⁻⁷⁴, reaching arbitrary depths at the expense of brain tissue displacement, and capturing imaging areas proportional to the device's small cross-section. Directly implanted CMOS image sensors (CIS) can also image at arbitrary depths⁷⁵⁻⁷⁹, and have an imaging area proportional to the widest side of the implant. Such implanted CISs have been shown to acquire neural activity, but not with single cell resolution in vivo^{75,79}. This is because they rely on contact imaging; only sources within about 10 microns of the imager are sufficiently resolved⁷⁸, and all sources more distant from the sensor contribute unfocused light. An open question is whether an optical probe can be made that images at depth from its widest side, yet achieves single-cell resolution.

I here show that it is possible, in theory, to take advantage of computational imaging strategies to invert the photonic signals acquired by an implantable CIS equipped with a light field mask (Figure 17). My strategy acquires high speed, 3-D distance-resolved, single cell neural activity signals. Rather than scanning a point—as in confocal or multiphoton microscopy—or having a single focal plane, my proposed system records data from the full 3-D volume in a single shot, without requiring moving parts. It does so without refractive optical elements, which can be sizable, and thus yields a high ratio of tissue volume imaged to tissue volume displaced. I also show that this system is robust against uncertainties

in the tissue's scattering and absorption properties, and can tolerate errors in the priors used in the image reconstruction process as well.

In this section, I outline a basic version of such a device and a simple reconstruction algorithm, simulate its performance with both current and potential future CISs and fluorophores, and examine metrics of neural signal fidelity. Our simulations indicate that it would be possible, using existing calcium imaging (a)

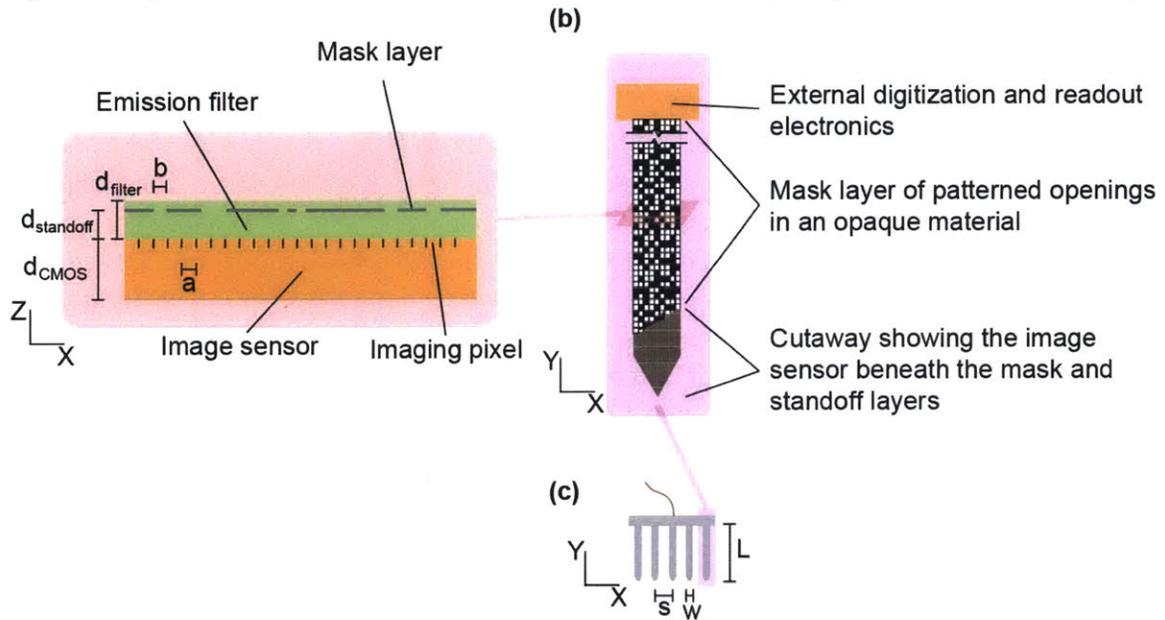


Figure 17 **a)** A schematic cross section of a single shank of an ILM probe. **b)** a schematic view of the wide, imaging side of the ILM shank. The probe length has been shortened for easy viewing. **c)** A view of a not-to-scale representation of what an array of probes might look like. Several of these comb arrays may be combined to image larger 3D volumes. See Zorzos et al.⁵³ for an analogous 3D array of light delivery probes.

reporters and CMOS pixel specifications, to acquire neural data with single-cell and single-spike resolution over relevant 3-D volumes adjacent to these probes, which we call implantable light field microimagers (ILMs). The reconstruction algorithm utilizes post-hoc anatomical knowledge to constrain the possible solutions to the correct set of neural signals. This process apportions all detected photons to neural sources, addressing the problem of out-of-focus light in software, unlike confocal or multiphoton systems that do this by the design of their hardware. Because such probes would be thin and acquire data from their widest sides, they could in principle be used to record at arbitrary depths or in multiple locations simultaneously while displacing a minimal volume of tissue.

1. Device architecture

Our device design (Figure 17) comprises one or more probe shanks that combine a thinned CIS, fabricated in the back side illuminated (BSI) geometry, in which the silicon is thinned from the back of the wafer to as little as 3.6 microns⁸⁰, and the photodiodes are exposed to light from this thinned back side. On top of the CMOS imager (Figure 17 a, orange), we deposit a lensless optical system. The lensless optics consist of a quasi-random (see II.A.2) patterned aperture mask (Figure 17 a, gray dashed line) of opaque and transparent regions—such as a $\sim 100\text{nm}$ thin metal layer with the metal etched away in a predetermined pattern, leaving a 50% open fraction—located $d_{standoff} \sim 5$ microns from the image sensor plane, with the separation provided by part of the optical filter layer. The lensless nature of this

system leads to optical path lengths within the device of only a few microns, in contrast to alternative solutions utilizing refractive optics^{81,82}. Even after adding an absorptive optical filter layer to block excitation light, of thickness $d_{\text{filter}} \sim 9$ microns⁷⁸ (Figure 17 a, green), the complete device is approximately 13 microns thick, on par with many kinds of implantable electrodes used today. The device shank is similar in geometry to dense electrode arrays⁸³ (Figure 17 b and c). Because the ILM contains only tens of kilopixels per shank, it can easily receive power from- and exchange digitized signals with- a computer using just a few flexible wires. This—combined with the small size and weight of the implant—would make ILM recordings compatible with freely moving behavior experiments.

2. Principles of device operation

This optical system does not generate a focused image, but rather the mask-sensor combination records information about the positions and incident angles of incoming photons. The intensity of light as a function of position and angle is known in computational photography as the light field. Each pixel detects light from a region of the light field determined by the mask: a given pixel can detect a photon if it passes through the position of an opening in the mask at an angle that reaches that pixel. Knowledge of the spatial and angular distribution of light from a source allows its location in three dimensions to be determined. Notably, this means there is no out-of-focus background light: There is no one focal plane, as each frame recorded by the imager records light field information about the full 3-D volume. This avoids the out-of-focus light problem faced by contact imagers and epifluorescence microscopes—and will allow the ILM to achieve single-cell resolution, even in vivo. This, combined with a lack of moving parts, means that frame rates can go much higher than imaging techniques that rely upon point- line- or plane- scanning to record 3-D volumes.

This philosophy of approach is shared with light field cameras and microscopes^{84–87}, but in our case, our quasi-random mask is equipped with a high open fraction (and thus high total transmissivity). This leads each pixel to record a random—but known, based on the mask pattern—superposition of several different segments of the light field. I also apply constraints on the solution—here, in the form of anatomical priors—to generate a single 3-D data set with no out-of-focus light at each time point. Past efforts have attempted to deconvolve the (synthetic) focal stack to estimate the locations and brightness of sources in (light field) microscopy⁸⁶. For conventional microscopes, the sources are convolved with a point spread function (PSF) that is roughly an ellipsoid with its waist in the focal plane. The ILM, instead, has a PSF that is the image of the source projected through the patterned mask. This PSF intentionally lacks translational symmetry in all dimensions—indeed, the mask is chosen to be quasi-random because that is one way to avoid aliasing problems possible with periodic masks. Each point in space having a unique PSF means the ILM reconstruction problem is a decomposition rather than a deconvolution, but both can be formulated as a system of linear equations.

In the generic case and with a reasonable voxel size—chosen, e.g. to keep the probability of a voxel containing two sources below a certain level, the number of voxels of the 3-D space to be imaged exceeds the number of pixels on the 2-D image sensor; this means the problem of inverting the measured intensity at all pixels into a 3-D dataset is underconstrained. Some form of outside knowledge or additional priors about the data are required to generate a unique solution to the reconstruction problem of determining the 3-D pattern of fluorescent emission from the 2-D image on the sensor. One of the most straightforward forms of outside knowledge is the anatomy, acquired either using a slow but high resolution imaging method (e.g., two-photon microscopy), or histologically after the end of the

experiment. Fluorescent molecules targeted to neuronal cell nuclei⁸⁸ allow for a sparse, punctate appearance - which helps constrain the problem by greatly reducing the number of voxels in 3-D space that could have emitted a signal photon. In many important cases—such as recording neural spikes—the problem further simplifies to determining the fluorescent power from each cell, rather than each individual voxel that contains part of the cell.

To enable mathematical analysis of the reconstruction problem, I frame it as follows: The patterned mask causes the fluorescence emitted from each voxel in 3-D space to project a unique pattern of light onto the image sensor. The image recorded by the sensor is the incoherent superposition of all the patterns from all voxels emitting fluorescent light. To reconstruct the fluorescent power of each source, this superposition must be decomposed into those unique patterns. This decomposition can be represented as a linear algebra problem (Figure 18 b). First we divide the space to be imaged into voxels. We then determine the pattern on the imager generated by light from each voxel (see II.C). This pattern is determined by the mask structure, light transport properties of brain tissue, and properties of the image sensor. We arrange the measured intensity on each pixel into a column vector Y , the fluorescent power of each voxel as a column vector X , and the pattern on the sensor projected from each voxel as columns of a matrix A (Figure 18 b). X is the quantity to be determined, Y is the measured data, and A is determined from knowledge of the mask and optical properties of brain tissue. The equation becomes

$$Y = AX.$$

Rather than solve for every voxel in the space, we apply our assumed knowledge of nucleus locations and instead solve only for each nucleus. Let M be the fluorescent power from each nucleus arranged as a column vector, and the columns of N be the pattern of light on the sensor from each nucleus—easily determined by summing the columns of A corresponding to the voxels containing that nucleus—to get

$$Y = NM.$$

This form of the equation will generally be overconstrained because the number of cells—and thus number of elements of the unknown M —is typically smaller than the number of pixels, which is the number of knowns in the column vector Y (Figure 18 c). The benefit of a spatially varying PSF caused by the mask is reflected in a smaller condition number for A than if there were no mask. Likewise, N has an improved condition number over A , as voxels near each other have relatively similar PSFs, so dropping voxels without sources reduces the number of highly similar columns in the matrix. Moreover, grouping all the voxels containing a given cell sums their corresponding (very similar) columns, further improving condition number.

This system is not designed to obtain diffraction limited images; instead, the ILM—and its reconstruction algorithm—are designed to achieve single-cell, single-spike resolution for hundreds of cells per ILM shank. As a result, we can choose voxel sizes larger than the diffraction limit. Using larger voxels also improves the condition number of the matrix A , rendering the equations more computationally stable, as well as smaller.

B. Results

1. Simulated device performance and performance limits

As a first exploration of what signals acquired by a prospective ILM might look like, we first simulated (see II.C) the fluorescence activity of neurons equipped with GCaMP6f and firing with a Poisson distribution with 4Hz mean rate. We then reconstructed traces of fluorescent activity (Fig. 3a) obtained via the simulated ILM (assuming a currently-available pixel size of 1.12 microns, and with a probe shank 80 microns wide and 13 microns thick—making our probe shanks similar in size to existing implantable silicon electrode arrays^{83,89}). This ILM is assumed to be combined with an excitation light delivery system (such as in^{52,53}) that—for simplicity in the current paper—is assumed to provide uniform illumination. We analyzed reconstruction quality with two different metrics: The correlation coefficient between the reconstructed and simulated traces (sometimes referred to as fidelity⁹⁰) (Figure 19b,c, 3-D simulations shown in X-Z projection), and the single-spike signal to noise ratio (Figure 19d,e). The noise was defined to be the root-mean-square difference between the reconstructed and simulated traces, and the single spike signal was the dF/F of a single spike, which we took to be 0.19 in our simulations, to match results of GCaMP6f *in vivo* in mouse pyramidal neurons⁶⁶. We used realistic scattering and absorption

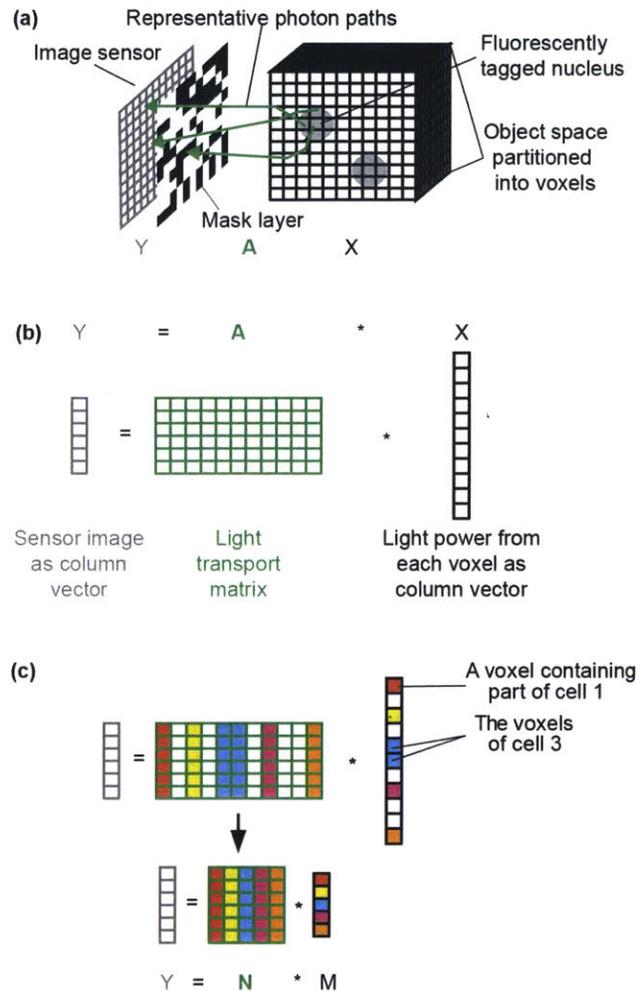


Figure 18 conceptualization of the reconstruction problem and the effect of applying constraints via anatomical knowledge

properties of brain tissue (Table 1). See Table 2 for the full list of GCaMP6f properties used, and Table 3 for the list of image sensor properties used. The voxel size was 4 microns on a side in all simulations.

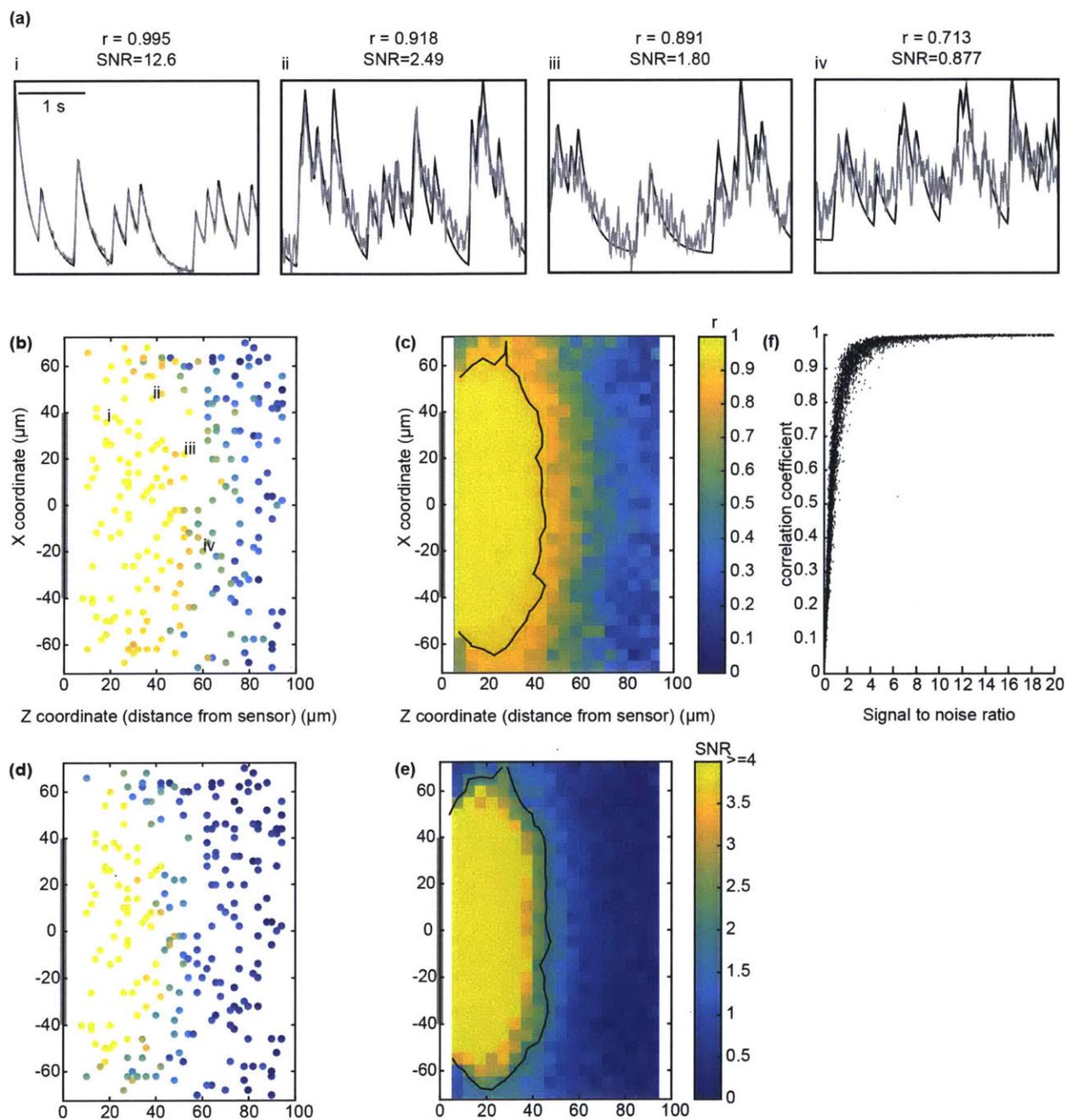


Figure 19 Simulation results for a representative ILM probe. **a)** Simulated GCaMP6 fluorescence traces (black) and their simulated reconstruction (gray), with associated correlation coefficients (r) and signal-to-noise ratios (SNR). Traces taken from neurons labeled in **b**. **b)** correlation coefficients for all neurons in a single simulation (Y axis suppressed). The ILM cross-section is represented by gray bar at left. **c)** Heat map of correlation coefficients as a function of position, averaged over 20 simulations. Contour corresponds to $r=0.9$. **d)** Same simulation as in **b**, but color now representing SNR. **e)** As in **c**, but color representing SNR and contour at $\text{SNR}=2$. **f)** Scatter plot of all neurons from the 20 simulations used in **a-e**, demonstrating the relationship between SNR and correlation coefficient.

We find good reconstruction (taken to be single-spike $\text{SNR} > 2$, roughly corresponding to a correlation coefficient of 0.9 for the given parameters; Figure 19 **f**) between the simulated fluorescent activity traces and the reconstruction for nearly all neurons up to ~ 40 microns in front of the probe (Figure 19 **b,c**; Figure 20 **a,b**). The Large range of incident photon angles accepted by the ILM enables some

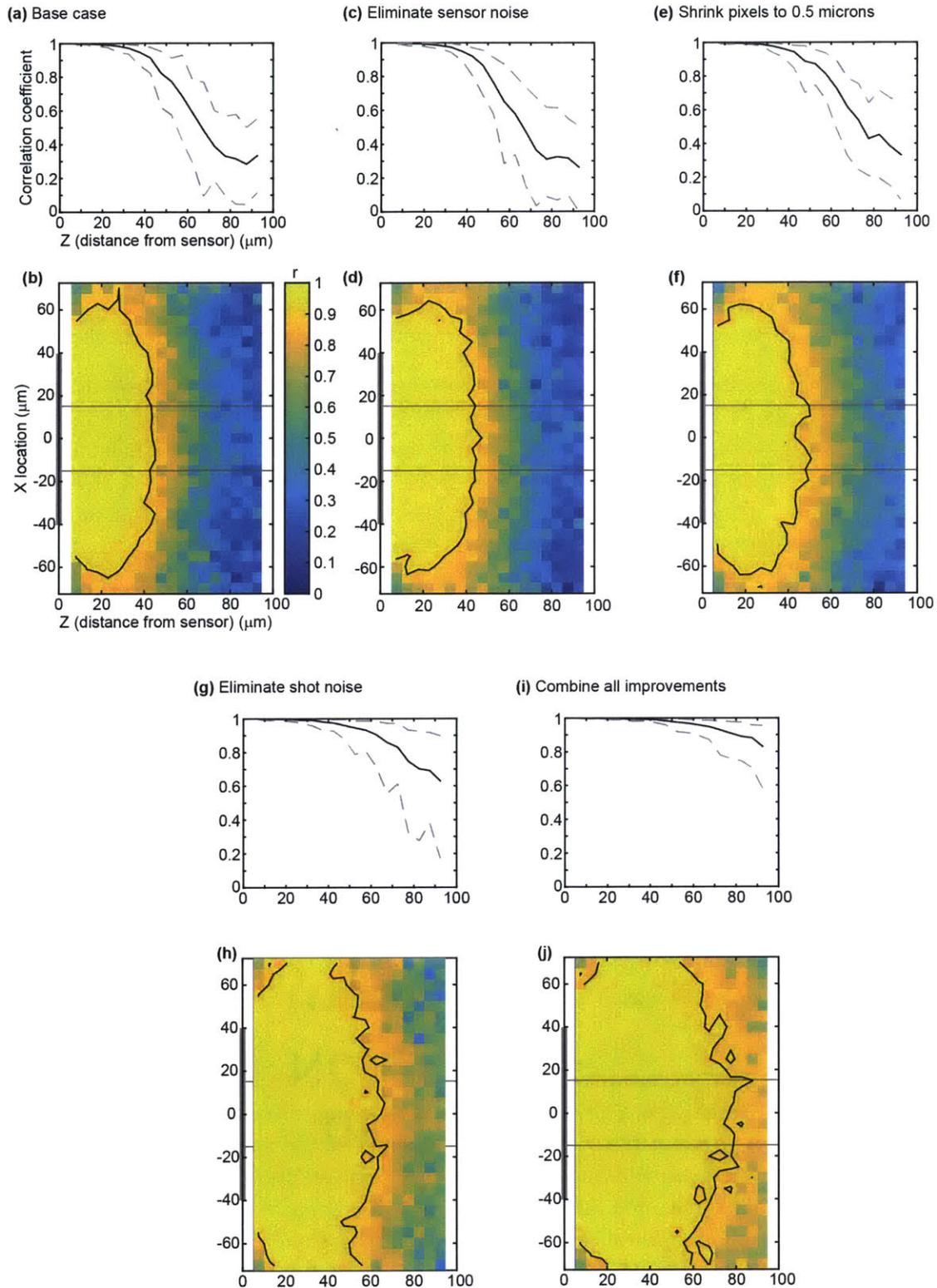


Figure 20 Determining sources of error. **a)** The realistic sensor case. Correlation coefficient as a function of distance from sensor, for cells centered in front of ILM probe (between light gray lines in **b**). Lines plotted for 5th, 50th, and 95th percentile. **b)** the full X-Z plot of data used to make **a**. **c,d)** As in **a,b**, but simulated to have no sensor noise (e.g., thermal, electrical, digitization, dark.) **e,f)** As in **a,b** but simulated with a sensor having a (currently unrealistic) pixel size of 0.5 microns. **g,h)** As in **a,b**, but simulated with no shot noise. This is the very bright fluorophore limit. **i,j)** As in **a,b**, but combining all of the prior improvements.

peripheral vision beyond the shank's width: The SNR contour exceeds 110 microns in width for our 80

micron wide simulated shank (Figure 19e).

For the section of probe shank simulated here (80 micron wide by 200 micron long active area), we find an average of 89 well-reconstructed neurons (1775 neurons over 20 independent simulation runs). Of those, an average of 77 (1534 neurons over the same 20 simulations) lie within the 110 by 200 by 45 micron volume in front of the probe. This is more than 90% of all neurons (1698 neurons over the same 20 simulations) in that volume. We define the well-observed volume to be this volume of space in which 90% of the neurons have an SNR greater than 2. This well-observed volume—and thus the total number of neurons that the ILM reconstructs well—depends on the total length of the probe shank. As with the width of the well-observed volume, it is slightly longer than the active length of the probe shank.

These results do not include any of the peripheral vision above and below the active area, as we did not simulate neurons there. For a neuron near the edge of the sensor, fewer photons are detected than for the same neuron located near the center of the detector. For these two reasons, extrapolating from this active area to larger active areas (i.e., longer or wider probes) should represent a conservative estimate for number of neurons detected. Thus, if we were to implant a single probe shank with 80 μm x 1 mm active area into the mouse cortex, we could realistically expect to observe about 383 neurons with SNR>2 in the well-observed volume and 440 well-reconstructed neurons total.

I also ran additional simulations taking more extreme values for several parameters. I did this to understand the sources of error in the reconstructions as well as to identify which specifications are most important when designing hardware, choosing fluorophores, and performing experiments. Since hardware and molecular reporters are constantly improving, these simulations may also suggest points in time in the future at which certain performance milestones may be met. The contributions of sensor noise, pixel size, and shot noise are presented in Figure 20.

Reducing the electronics noise to zero and increasing the digitizer to 14 bits—making digitization level spacing substantially less than 1 electron and thus negligible—(Figure 20 c, d) causes a slight improvement in accuracy of reconstruction as measured by the correlation coefficient. This indicates that typical electronic noise and digitizers commonly seen in off-the-shelf CMOS imagers are good enough for use in a successful ILM; no cooling of the sensor or high resolution digitizer is required. As will be demonstrated below, the reason is likely because the pixel size and especially shot noise provide stricter bounds on ILM performance.

Shrinking the CMOS pixels to 0.5 microns—resulting in the total count of simulated pixels increasing from 12,888 to 64,000—while leaving other parameters at their original values resulted in a modest performance increase (Figure 20 e, f). Some improvement with reduced pixel pitch is to be expected, given that the lensless nature of our probe does not allow for magnification. In particular, it improves the angular resolution of the recorded light field. Two emitters near each other but distant from the ILM probe project very similar images through the mask onto the sensor, and thus smaller pixels can resolve the difference between them better than larger pixels. In the linear algebra conception of the problem, smaller, more numerous pixels render the problem less underconstrained. More importantly, they also improve the condition number of \mathbf{A} by rendering the columns corresponding to nearby voxels less similar. 0.5 micron pixels is an extreme value, not currently commercially available, but demonstrates the extent to which a pixel size reduction will, all other things being equal, improve performance.

Finally, eliminating shot noise (Figure 20 g, h) from the simulation resulted in a large performance improvement. Given that shot noise scales as $1/\sqrt{N}$ where N is the average number of detected photons, eliminating shot noise corresponds to the limit of very bright fluorescence. This shows that even an implantable imaging system that collects light over a large solid angle benefits from brighter fluorescent signals. This stems from the fact that more distant neurons contribute far fewer photons to a given pixel than nearby neurons. While a source's signal on a pixel scales with the number of photons from that one source, the shot noise is proportional to the square root of photons from *all* sources detected by that pixel. Accordingly, fluorophore brightness, excitation light intensity, and frame rates will be important parameters with the use of ILMs.

Combining all three of the above optimization explorations—eliminating sensor noise, eliminating shot noise, and shrinking the pixels to 0.5 microns—(Figure 20 i, j) leads to excellent ($r > .9$) reconstruction of neural activity for neurons out to 70 microns. Note that this indicates that our reconstruction method can perform well in the presence of photon scattering by brain tissue, as tissue scattering and absorption were included in all simulations. This demonstrates that we have identified major contributors to error in the reconstruction of neurons near the sensor, at least within the bounds of our model of the physical process of light generation, propagation, and detection. The simulated reconstruction is imperfect for two reasons: One is that the pixel size is still finite; as noted, two distant voxels near each other will have very little angular separation between them, and once the shot noise is eliminated, the poor condition number of the matrix \mathbf{M} due to such voxels becomes apparent. The second one is computational. As noted in I.B.4, because any Monte Carlo photon model simulates a finite number of photons, there will be Monte Carlo shot noise. This problem is exacerbated for distant sources and small pixels. The number of photons hitting a pixel is proportional to pixel pitch squared, so halving the pitch requires four times the simulated photons. The number of photons hitting a pixel (from a small, isotropic source, as these are) scales with distance via the combination of $1/r^2$ falloff and the exponential falloff of the effective penetration length of the fluorescent light. That said, the Monte Carlo shot noise (and hence simulated pixel pitch limits) is purely computational and with enough compute power, this effect can be pushed arbitrarily low.

2. Temperature changes caused by the device

Effects on brain temperature should be considered any time a device is implanted in the brain. Silicon electronics can generate extremely high heat power densities, so heating is an important concern with any probe that involves active electronics. On the other hand, cooling is also a frequently overlooked issue: silicon and metals—dominant components of many implants, including ILMs—are good thermal conductors and will wick heat away to be dissipated outside the brain, cooling nearby brain tissue from body temperature towards ambient temperature. We find that these competing effects can be balanced to keep all brain tissue within ± 1 C of body temperature for a wide range of realistic probe geometries (Figure 21). The heat generated need only approximately match the heat dissipated—via convection and radiation off the external electronics in our model. Thus for a wide range of probe geometries, a frame rate—corresponding to a particular heat generation rate—can be found that keeps temperature changes smaller than one degree.

One typical example: a probe targeting a deep structure with shank length 5.1mm and pixels covering the deepest 1mm with external electronics having a cooling area of 8mm^2 running at 110 Hz has a

maximum increase in brain temperature of 0.85 C near the active pixels and a maximum decrease in brain temperature of <.01 degrees, where the probe enters the brain tissue through the skull. Adding a simple active cooling technique—such as a small Peltier element to keep the probe's external

temperature fixed, the brain can be kept within ± 1 C over a much wider range of frame rates— including avoiding excessive brain cooling while the probe is not running (Figure 21 b,d,f, solid lines).

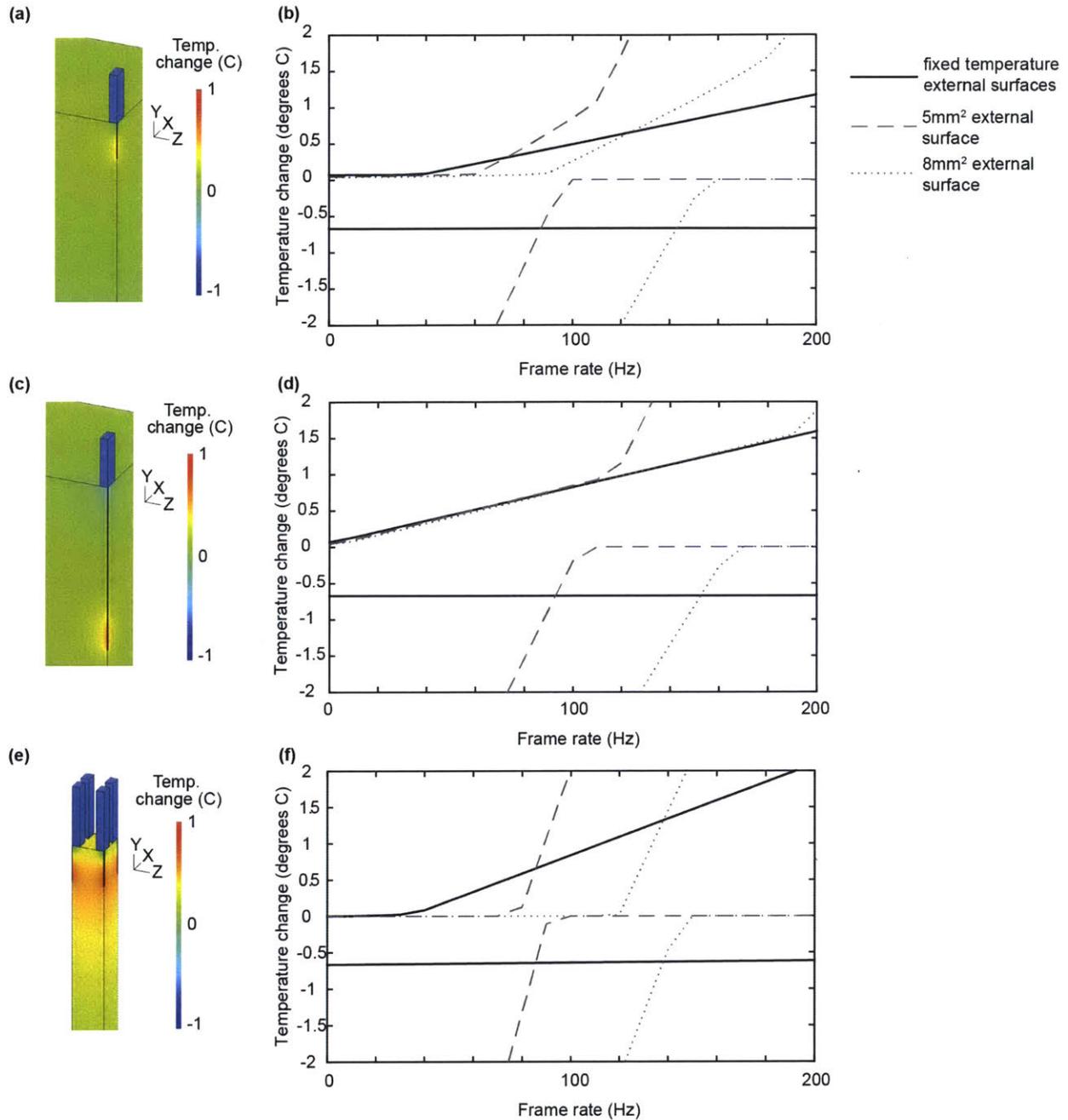


Figure 21 Simulated temperature changes caused by ILM probes. **a**) A 1.1mm probe with lowest 1mm active. Cutaway through the center of the probe in the X and Z axes showing temperature difference from intact brain in probe and brain. **b**) Plots of maximum positive (upper lines) and negative (lower lines) temperature change anywhere in the brain, as a function of probe frame rate. **c,d**) As a,b for a probe 5.1mm long with the lowest 1mm active. **e,f**) As a,b but for an infinitely repeating tiled array with 1mm probe pitch of probes 1.1mm long with lowest 1mm active. Axis indicators are 1mm long.

3. Sensitivity to input parameters

Calculating the light transport matrix A requires knowledge of the mask and optical properties (i.e., absorption coefficients, scattering coefficients, and phase functions) of the brain tissue. Because the optical properties of brain tissue are often only known approximately (see I.B.3), we tested the effect of having errors in the scattering coefficient. We chose, as a test case, an extreme example: generating A assuming a perfectly transparent brain, then using that A to reconstruct ILM images simulated with realistic scattering. Even with such a naive assumption, the performance was not degraded (Figure 22 a). For this particular mask pattern and sets of neuron locations, it was insignificantly improved: The A calculated using optical parameters matching the simulated sensor images detected 1775 neurons with $\text{SNR}>2$ over 20 simulation runs of an 80×200 micron sensor, while the zero-scattering A reported 1868 neurons with $\text{SNR}>2$ over 20 equivalent simulation runs. This indicates that over the distance scales relevant to ILM usage, the light transport matrix is dominated by the ILM hardware and the intensity falloff as a function of distance; the impact of attempts to incorporate scattering into the light transport matrix depend on the properties chosen for the mask (not shown), but is not generally large. This by no means implies that scattering does not have an impact on imaging inside the brain; it certainly does. The result of Figure 22a indicates only that the light transport matrix is not dominated by scattering/absorption over these length scales, and attempts to include scattering can have mixed effects—while each column of A might be more accurate, they become more similar to each other (as scattering smears the light across more pixels) and the condition number of A is reduced. Thus, the reconstruction becomes more sensitive to any source of noise.

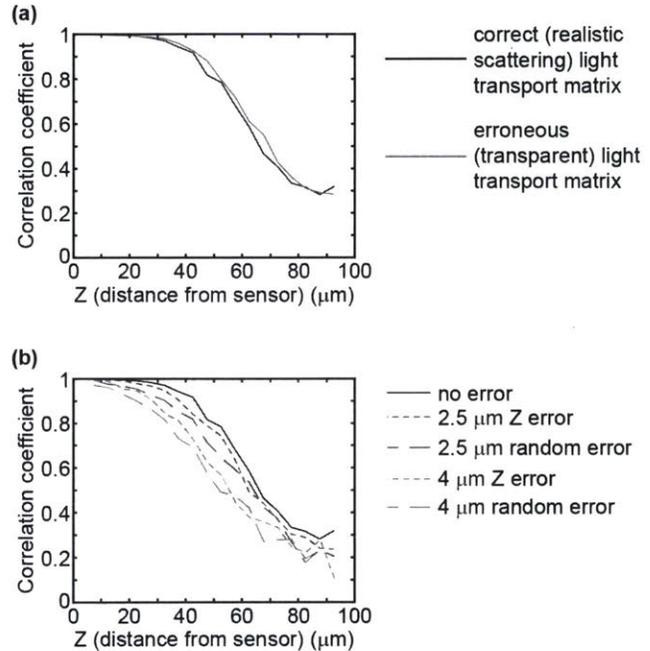


Figure 22 Robustness to uncertainty in optical properties and anatomical priors. **a)** Reconstruction of neural activity simulated with realistic brain tissue when the reconstruction uses a realistic light transport matrix (black), or a manifestly wrong transparent light transport matrix (gray). Highly accurate optical properties of brain tissue are essentially irrelevant. **b)** Reconstructions using an M matrix generated with correct anatomical priors and with either systematically incorrect locations in Z , or per-neuron errors randomized in direction, of the listed magnitudes. Accuracy of priors is important, but not overly sensitive to relatively large (on the scale of neurons) errors.

Because our proposed system uses priors to constrain the reconstruction and generate the matrix M , we also evaluated the sensitivity of results to errors in the priors—in this case, errors in the spatial information of cell nuclei locations relative to the ILM (Figure 22 b). A uniform error of 2.5 microns—the radius of the simulated nuclei—in the Z direction for all neurons results in a modest decrease in performance, with the total number of well-reconstructed neurons expected on an 80 micron by 1mm probe falling from 440 to 374. The formerly well-reconstructed volume now has 80% of its neurons with $\text{SNR}>2$, down from 90%. Position errors of 2.5 microns in independent random directions for each neuron result in a slightly worse decrease in performance, though this scenario is likely unphysical, as most position errors are likely to be (potentially nonlinear) shears, translations, scales, etc. and thus a

relatively smooth function of initial position. Increasing the error magnitude to 4 microns in the Z direction yields an expected well-reconstructed neuron count of 262 and the formerly well-reconstructed volume now has 60% of its neurons with SNR>2.

C. Methods

We used a three stage process to simulate the data recorded by the ILM, followed by a reconstruction step where we used the simulated data to calculate the physiological activity of each cell. In this section, we first briefly outline each stage, before discussing each in more detail below. In the first stage (II.C.1), we used a Monte Carlo (MC) code based on the one used in³⁶ to simulate the propagation of light from a fluorescent nucleus. This shared the large majority of its code base with the MC code described in I.B. As noted earlier, fluorescence was simulated to be localized to the nucleus, modeling the use of a technique such as in^{87,88}. The MC code simulates propagation from the fluorescent nucleus through the brain tissue to the ILM mask plane. This code records the position and angle of each photon hitting mask plane, which is the light field resulting from that nucleus's fluorescence. If a cell in homogeneous gray matter is moved parallel to the mask plane, its light field at the mask plane is simply translated; rather than calculate a light field for every position in the x-y plane, we calculated it at one x-y location and added a displacement. On the other hand, if a cell's distance from the mask plane changes, the angles and relative positions of the photons hitting the mask plane will change, so we calculated a light field for each of several different distances between the cell and mask plane.

The second stage of the simulation (II.C.2) models the fluorescent activity to be observed: The simulation randomly positions neurons in the volume to be simulated. It generates spike times for each neuron randomly from a Poisson distribution with a mean firing rate of 4Hz, and the resulting fluorescence intensity of each neuron at each frame time are generated, based on properties of GCaMP6f⁶⁶. We use this brightness and location data for each neuron in conjunction with the light fields calculated in stage one to determine the total light field due to all neurons at the mask plane.

The third stage (II.C.3) takes the light field computed in stage two and converts it to an image recorded by the ILM sensor. It first rejects photons blocked by the opaque mask layer and incorporates refraction by the absorption filter to determine on which pixel each photon impinges. It then uses the CMOS sensor's photoconversion efficiency, read noise, and digitization noise to generate the ILM image as recorded by the sensor. We determined diffraction effects by the mask to be present but minimally important to our recorded images, thus allowing us to use the ray optics formulation of the Monte Carlo code and avoid the massive computational burden of full wave simulations.

The fourth stage (II.C.4) first generates the light transport matrix **A** by using a similar process for the light from voxels that was used for neurons in II.C.1 and II.C.3. The generated ILM image from each voxel form the columns of **A**. The reconstruction algorithm uses each simulated ILM image of brain activity to calculate a fluorescence intensity for each neuron at the time the image was recorded. Reconstructing the full series of sensor images thus yields fluorescence traces for each neuron in the simulation. These reconstructed traces can be quantitatively compared to the neuronal fluorescence traces as simulated in the second stage to determine reconstruction correlation and signal to noise ratio.

1. Simulating light propagation in brain tissue

We used our MC code to simulate one neuron fluorescing at each of a range of distances from the plane containing the ILM's mask (i.e., at a range of positions along the z axis), to account for the light field changing with z position. We modeled the cell nucleus as a 5 μm diameter sphere. The distances simulated were at 2 micron intervals, starting at 6 microns—any cells closer would have cell bodies intersect the ILM—out to 94 microns—so as to leave the entire cell body safely within the 150x200x100 micron simulated volume. We simulated the brain tissue as uniform gray matter with photons of wavelength of 532nm—near the peak of the GCaMP6f emission—propagating through it. See table 1.A for the parameters used. Each MC photon's starting location was randomly chosen from within the 5 micron diameter sphere of each cell nucleus, and its direction was randomly chosen from an isotropic distribution.

The code propagated each MC photon for a random distance chosen from a Poisson distribution with a scattering coefficient of 23.3 mm^{-1} ³⁶. The code then selected a new direction by sampling from the Gegenbauer kernel phase function with a scattering coefficient $g=.949$. Rather than probabilistically determining if the photon was absorbed or not, we instead introduced a weight that equals the probability that the photon has not been absorbed. This weight was updated before each scattering event, based on the distance travelled since the last scattering event and the absorption coefficient, taken to be 0.942 mm^{-1} . Using this weight factor rather than probabilistically absorbing the photons allowed the MC model to generate good statistics with far fewer simulated photons than would otherwise be required. The code simulated each photon until it reached a boundary of the MC simulation volume, at which point its position, direction, and weight is recorded. The simulated volume was a 4mm cube, much larger than the 80 by 200 micron simulated sensor, so as to avoid edge effects breaking the translational symmetry in the ILM mask plane. The output of this code is a file for each z position of the source containing the final position, direction, and weight of each simulated photon. See I.B for more details on this code.

We ran 4×10^7 Monte Carlo simulated photons for each neuron. The finite number of simulated photons introduces an additional "Monte Carlo shot noise"—purely an artifact of the simulation, not present in the physical system—that makes all simulated reconstructions slightly worse than equivalent real-world reconstructions would be, given that I also add the real photon count shot noise to the model in II.C.3. 4×10^7 was chosen as a number that kept this artificial error small for all but the dimmest regions of the light field, while keeping computation time reasonable.

2. Simulating neuronal activity and resulting fluorescence

The second stage of the simulation begins by randomly distributing neurons in the volume to be simulated with a default density of 10^5 neurons/ mm^3 , roughly the density of neurons in the mouse cortex⁹¹. We simulated a volume 150 microns wide by 200 microns long by 100 microns deep. We simulate the spiking of each neuron as following a Poisson distribution with a mean spike rate of 4 Hz. We simulate the fluorescence intensity of each spike as a linear rise to a peak dF/F of 0.19 over 45ms, then an exponential decay with a half-life of 142ms⁶⁶. The fluorescent waveforms of multiple spikes were modeled as adding linearly, as at this spike rate the GCaMP6f never approached calcium saturation, i.e. the dF/F levels found in simulation were far lower than dF/F at saturation, as would be expected physiologically for a 4Hz average firing rate. We took baseline fluorescence F_0 to be 8.9% of

EGFP fluorescence, based on comparison of EGFP and GCaMP baseline fluorescence in under similar conditions in neurons^{66,92,93}. See Table 2 for a complete list of parameters used in this simulation.

An important aspect of any fluorescence imaging system is excitation light delivery. We do not address that here, as excitation light delivery is a relatively mature—though rapidly advancing—field with different options tailored to different experimental conditions. Instead we focus entirely on evaluating the performance of the novel component of the system, the ILM fluorescent emission recording system. We therefore simulate the use of a uniform excitation illumination with a fluence of $31\text{mW}/\text{mm}^2$ at 465nm for simplicity.

3. Simulating ILM images

We calculate what the image on the ILM from a single neuron would be by loading the light field data generated in step 1, culling all photons blocked by the mask, refracting and propagating the remaining photons through the absorption filter, and assigning them to the pixel they hit based on their x-y coordinate at the CIS plane. We then multiply each single neuron's image at a time t by the intensity of its fluorescence at t and the fluence of the excitation light—taken to be uniform through the volume simulated. The total pattern on the ILM sensor at time t is then the sum of all of the individual neuron images at time t . We then add Shot noise to the pattern, convert from photon count to generated electron count, add CMOS electronics noise, cap the electron count at the well depth, and digitize the signal to the CIS's ADC bit depth. See Table 3 for CMOS imager parameters, which were taken from^{94,95} so as to be realistic values for current, off-the-shelf pixel technology. We simulate back side illuminated imagers because they offer improved light collection, improved sensitivity at high incident angles when compared to front side illuminated imagers, and already have a thinning step that leaves the sensor thin enough for tissue implantation.

We simulate an ILM shank 80 microns wide and 200 microns long. We chose an 80 micron width as it is a size comparable to or smaller than many implanted probes used today. Both narrower and wider shanks also work as well, with corresponding decreases and increases in width of view (not shown). We chose the 200 micron length to make the simulations conveniently fast, for rapid iteration of simulation parameters. In practice the active shank length could—and often would—be much longer, on the order of millimeters or more. Edge effects at the ends of the 200 micron length only serve to reduce the predicted imager performance, as we have effectively half the pixels to image a neuron near the ends of the simulated segment, compared to if the probe shank were longer.

Diffraction is an important concern when dealing with apertures on the same order as the wavelength of interest, which is the case for an ILM. We simulated diffraction through 2.6 micron square apertures—the size we used for individual mask elements in the simulations presented—at various angles to evaluate the effect of diffraction. We were careful to write the code such that it makes no paraxial approximations, as without any lenses or apertures, the ILM has no restriction on possible incident angles. We also made no far-field approximations, as the image sensor is set to be only 5 microns from the mask layer. For incident angles up to approximately 45 degrees, diffractive focusing makes the 0^{th} order spot full-width-at-half-maximum smaller than the 2.6 micron aperture it passes through and does not substantially alter the location of the spot (Figure 23). For propagation angles above approximately 60 degrees, the 0^{th} order spot is broadened and displaced from where it would be expected according to ray optics. It is possible to generate the light transport matrix \mathbf{A} using wave simulations of propagation

through the mask to the sensor to incorporate these diffractive effects, but it would be computationally expensive with our hardware. A first order correction for diffraction could perhaps be obtained in a feasible timeframe with modern hardware by generating a lookup table of diffraction patterns. MC models would still be used to generate the light field at the aperture mask, then the diffraction patterns would be extracted from the lookup table and weighted based on the calculated light field at each aperture, then superimposed to make the final sensor image. The naïve sum—without including phase—works as long as the detected light is incoherent, which conventional fluorescence is.

We, however, used ray optics in our simulations of device performance, and there are reasons to expect the impact of highly oblique, severely diffracted incident light to be minimal: 1) The exposed area of an aperture to light is reduced by a Lambertian factor of the cosine of the incident angle: 75% of the photons reaching the sensor have an incident angle less than 60 degrees if the fluorescent sources are uniformly distributed in space. The percentage is even greater if the fluorescent sources (or excitation

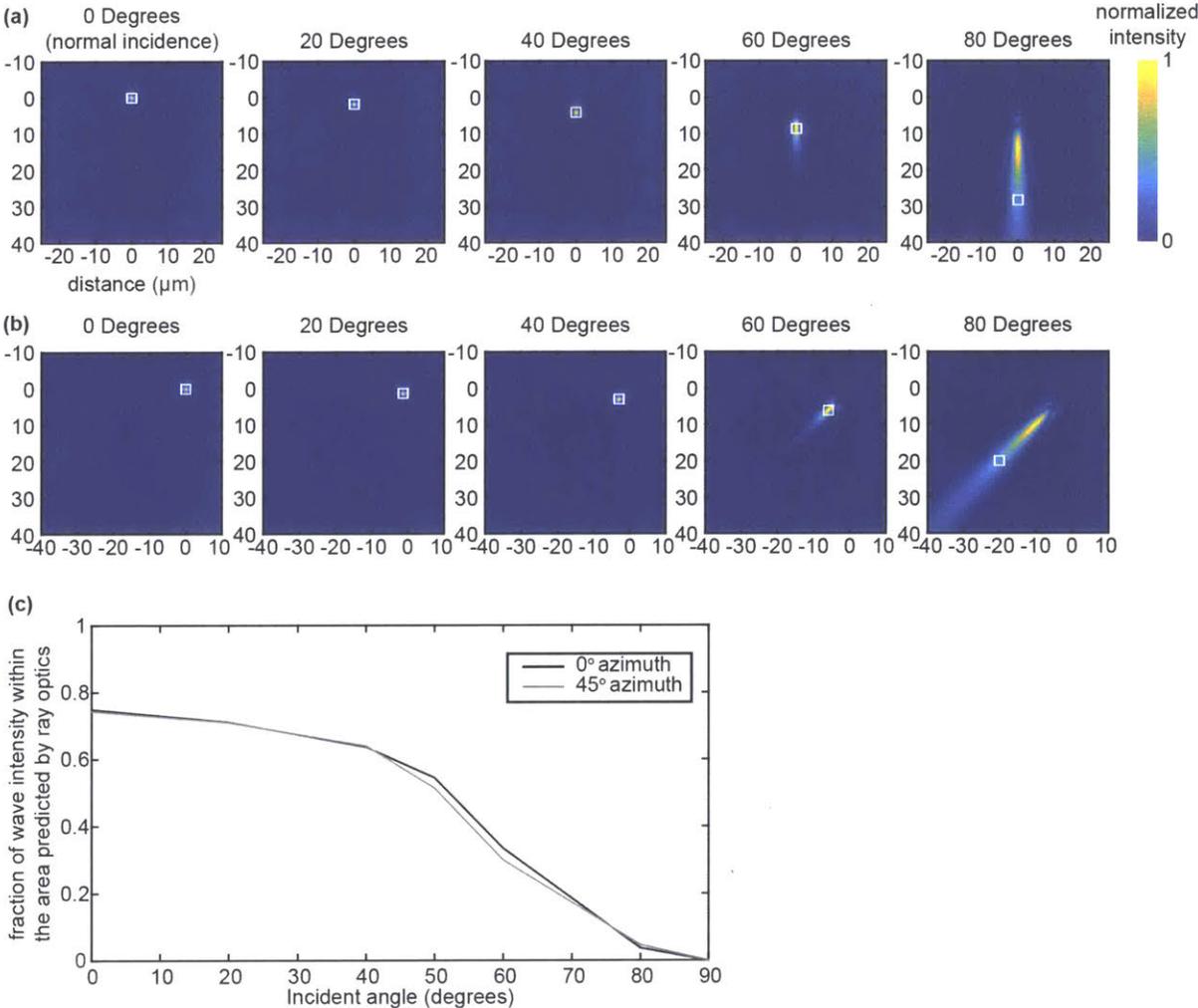


Figure 23 Evaluating the impact of diffraction. **a)** diffraction pattern at the sensor through a single mask aperture hole. The white square is the location predicted by ray optics. Plane waves are incident with the labeled angle, measured from normal. **b)** as **a)**, but with the incident light having an azimuthal angle of 45°. **c)** The fraction of total light power falling within the area predicted by ray optics as a function of angle.

light) is localized to any degree in front of the probe. 2) Due to the patterned mask, the diffuse, streaked pattern of badly diffracted light will not resemble any source's image on the sensor—i.e., the Cartesian distance between badly diffracted light and a source's predicted image is large. This makes crosstalk with other sources' images—and added reconstruction error—minimal. We also note that when an ILM is built, the actual sensor image due to light sources at each position in front of the probe could be experimentally measured, including all physical effects before implanting the ILM probe shank, rather than simulated on computer. As noted, this experimentally measured light transport matrix wouldn't precisely mimic in vivo conditions, even though relatively realistic tissue phantoms exist. However, see Figure 22a for the (relative lack of) impact of calculating A outside of the scattering medium of the tissue. These measurements would also detect and correct for any flaws in the device as actually fabricated—such as dead pixels or mask pattern flaws—by empirically measuring actual illumination patterns recorded by the image sensor.

4. Reconstructing physiological activity

Fluorescent light emitted from any point in the imaged volume propagates through the brain tissue and the mask, creating a particular pattern of illumination on the sensor. For computational purposes, we divide up the imaged volume into 4 micron cubic voxels and calculate the illumination pattern on the CMOS imager from each voxel. The image on the sensor from an arbitrary pattern of fluorescent emission in the volume can thus be calculated as the superposition of the images from each fluorescing voxel, multiplied by the intensity of fluorescence in that voxel. This can be represented in a linear algebra formulation as $Y = A * X$ (see Figure 18). Y is the sensor image arranged as a column vector, X is the fluorescent intensity in each voxel, and A is the light transport matrix that maps the pattern of emission X onto the resulting sensor image Y . Each column of A is the sensor image of fluorescence from a single voxel. We calculate the columns of A using the same process as described in “simulating ILM images”, but the Monte Carlo simulations use 4 micron cubes that correspond to the voxel size rather than the 5 micron spheres used to simulate nuclei. No noise is added to the simulations, as we are generating the “ground truth” light transport matrix, rather than simulating a realistic experimental measurement.

The reconstruction algorithm first utilizes post hoc knowledge—and the localization of fluorescence to the nucleus—by combining the elements in X that correspond to voxels that include a particular source. This results in a column vector M with one entry per source. The corresponding columns of A are also combined into a matrix N , yielding an overconstrained problem $Y = N * M$. This is then solved for M using the QR algorithm via Matlab's `linsolv()` function. This algorithm is repeated at each frame time to generate reconstructed fluorescent traces for every cell. The reconstructed trace for a cell i is simply $M_i(t)$.

Noting the impact of shot noise (see Figure 20g, h)—which has a white spectrum—and indeed observing much of the noise to be high frequency (see Figure 19a), we applied a simple 5 sample moving window filter (33ms wide)—similar to ⁹², which used 100ms moving window—to the reconstructed traces. This improved the correlation, especially for more distant neurons (Fig. 8). Wider moving windows resulted

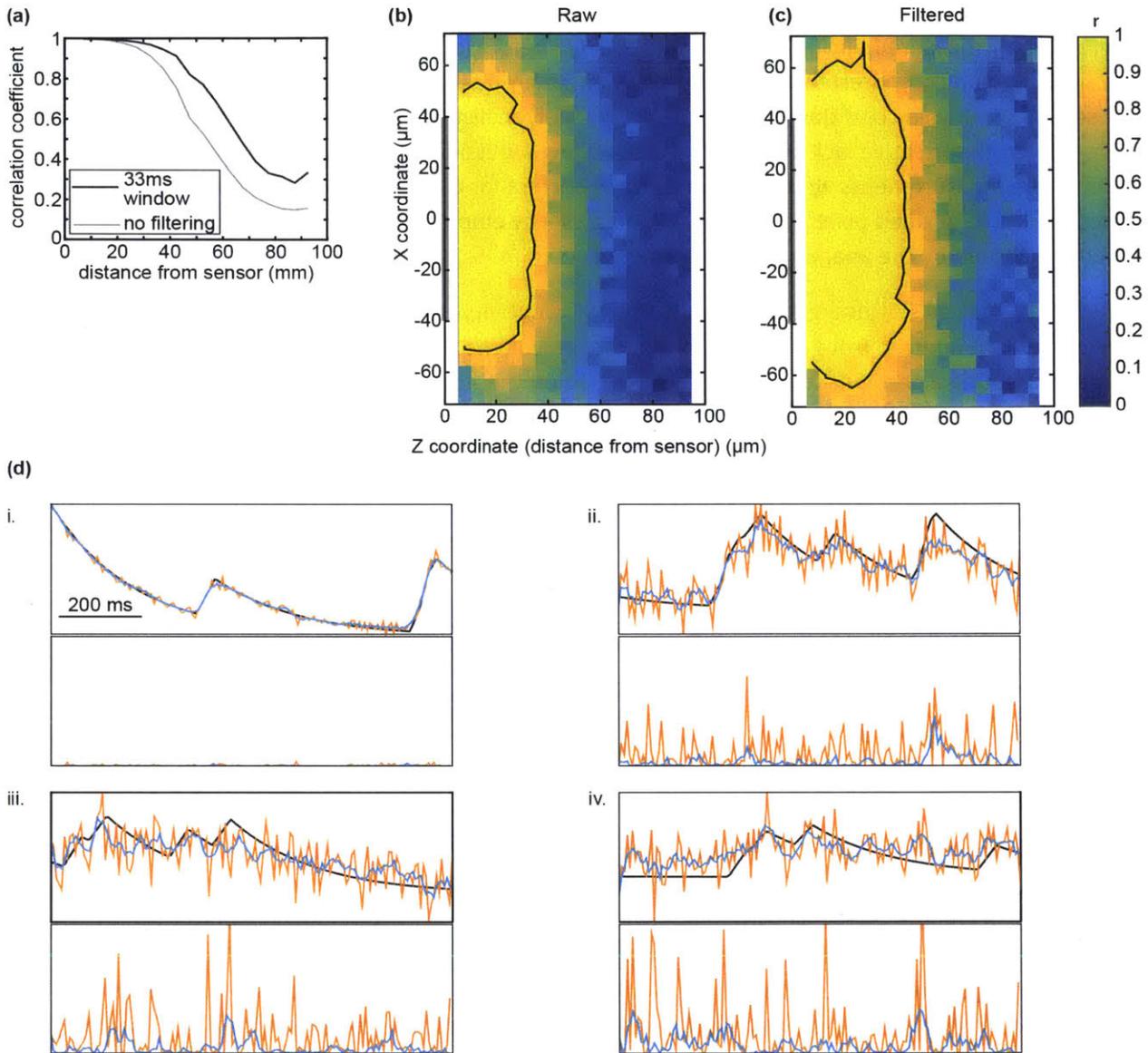


Figure 24 even basic moving average filters improve performance. **a)** Comparing filtered vs. unfiltered performance as a function of distance from sensor plane. **b,c)** correlation coefficient as a function of X-Z position, averaged over 20 simulations with and without filtering. Top panels: black is the fluorescence as simulated, blue is reconstructed trace after filtering, orange is raw reconstructed trace. Bottom panels: squared error of the filtered reconstructed traces (blue) and raw reconstructed traces (orange). i, ii, iii, and iv correspond to the same traces as in Figure 19a. However, only the first second of each trace is shown for improved clarity.

in marginal further improvements in correlation for more distant neurons, at the cost of slightly worse correlation for nearer neurons, as the filter impinges on the higher frequency components of the real fluorescent signal (Not shown).

5. Simulating Heating

To simulate the heating of the brain by the ILM's active electronics (Figure 21), we used a finite element model (Comsol 4.3b⁹⁶) of a block of brain tissue, the implanted ILM shank—simulated for the thermal model as a silicon core 10 microns thick and 80 microns wide, with the imaging surface covered by a 9 micron absorption filter consisting of a polymer impregnated with an absorptive dye—and the external ILM electronics. See tables 2 and 3 for the values used in the simulations and the references from which they were drawn. The top surface of the brain tissue is assumed to be perfectly insulating: Because of uncertainties in heat conduction through dura, skull, skin, and hair, we make heating's worst-case assumption that no heat can escape through that surface. Any heat conduction through this surface serves to mitigate temperature increases in the brain. The brain tissue itself was modeled as a thermal conductor with a blood flow heat term inversely proportional to the tissue's deviation from body temperature, as in⁹⁷.

For the single probe geometry, we exploited mirror symmetry in the planes normal to X and Z, simulating only one fourth of the probe and applying insulating boundary conditions (appropriate for the heat equation) on those symmetry planes. The block of brain tissue simulated was a 10mm cube (effectively a 40x10x40mm box, after symmetry unfolding) with insulating boundary conditions. The very large brain volume simulated ensured that boundary effects of the distant sides and bottom of the simulated volume were minimal, as confirmed by parameter convergence studies²⁵ showing even larger volumes yielded indistinguishable results. By using the insulating boundary condition—which will cause the temperature to be higher than if the heat were allowed to conduct farther than the 10mm cube—we again ensure that any boundary effects that did exist would lead to pessimistic results for how much heating will occur. For clarity, only a portion of the simulated volume near the probe is displayed in Figure 21.

For the repeating probe geometry, we use a 1x10x1mm box of brain tissue with an insulating bottom boundary and periodic boundary conditions on the sides. This makes the simulated volume one unit of an infinitely repeated probe array covering an entire brain's surface. The brain tissue was modeled as a thermally conductive medium with a constant metabolic heat source and a term inversely proportional to temperature that simulates active heat transport (via blood flow, primarily), as in⁹⁷. The equilibrium temperature of the brain tissue with the parameters used is 37.1C.

A CIS dissipates only a fraction of its power at the pixels themselves; the remainder is generated by digitizers, line drivers, timing circuits, etc. and can be located on the part of the ILM probe external to the head. Relatively few papers on CIS technology break down the power consumption of these various components, but published work indicates that 17%⁹⁵ or less⁹⁸ of the total power is generated at the pixels, with that percentage decreasing with increasing framerates⁹⁸. We assumed 17% of the total power was generated uniformly in the part of the silicon probe shank covered by CMOS pixels, while the remainder was generated uniformly in the part of the ILM external to the brain. Our model included either convective and radiative cooling into the environment from that portion of the probe external to the head, or a fixed temperature for the surface of the external electronics, which could be achieved by a simple feedback circuit with a Peltier element and a temperature probe mounted on the external electronics. We used specifications from a commercially available CMOS image sensor⁹⁴ to calculate realistic heat generation values. The OV13850 is a 13.2 million pixel imager that uses a total power of 199 mW when operating at 30 Hz. This is a total energy of 0.50 nJ/pixel/frame to record and read out

one pixel. 17% of that is .085 nJ/pixel/frame generated at the pixel. The total power dissipated within the active portion of the shank is this value multiplied by the total number of pixels and the framerate.

We simulated several shank geometries: Figure 21a and b use a single shank 1.1mm long (i.e., extending from the surface of the brain to a depth of 1.1mm) with pixels covering the bottom 1mm. Figure 21c and d use a single shank 5.1mm long with pixels covering the bottom 1mm. The total pixel counts were thus 63,400. Figure 21e and f use a repeating probe geometry, with probes placed on a 1mm grid across the surface of the brain. Each probe is 1.1mm long with pixels covering the bottom 1mm and again having 63,400 pixels. By using insulating boundary conditions—which, for the heat equation, is identical to periodic boundary conditions—we simulated the limit of an infinite array of probes covering an arbitrarily large brain. This is also a pessimistic limit for calculating maximum heating.

D. Discussion

We set out to determine realistic performance expectations for an ILM system. Our findings indicate that an ILM built with current technology and using current fluorophores would indeed allow recording at depth of many cells with single-spike, single-cell precision. The simulation data presented here show that this technology could open up new regimes of optical recording of physiological activity for neuroscience—and likely other areas of experimental biology, as well.

While the ILM concept as simulated in this paper uses only existing commercialized technology and fabrication techniques, it would require a custom layout for its CMOS imager probe shanks using a relatively modern fab process. The custom lithography mask, minor retooling, and the interruption of throughput of large-batch commercial product makes the first wafer of ILM probes relatively expensive, even though additional wafers would have low marginal cost (often less than \$20/unit for high production volume CMOS imagers, even high quality ones). The expense of the initial wafer will drop considerably as custom/small batch fab shops acquire the necessary lithography and wafer manipulation equipment. The migration of manufacturing processes from global-scale industrial foundries to custom fab shops is always ongoing, and small fab shops can already produce imagers with nearly small enough pixels on-demand, with BSI processes already on the horizon.

For fluorescence imaging of neural activity, we see three main categories of ways to detect the fluorescence photons: allow the photons to pass through brain tissue to an external detector, allow the photons to pass through implanted optical elements—such as optical fibers—to an external detector, or detect the photons inside the brain and relay the data out—most simply via electrons. The first has the advantage of not displacing any brain tissue at the expense of depth limits and degraded signal, the second has the advantage of reaching any location in the brain at the expense of displacing brain tissue, and the last can reach any location while displacing little brain tissue relative to implanted optical elements. In both of the latter cases, the displaced tissue volume is proportional to the probe's cross-section. However, implanted optical elements' imaged volume is also proportional to the probe's cross section, while the ILM's imaged volume is proportional to the probe length multiplied by the wide side of its cross section, yielding a much better damaged to imaged volume ratio.

A major challenge with a directly implanted detector is resolving sources at the single cell level. Our light field mask and associated reconstruction algorithm allows this by apportioning all detected photons to sources, leaving no “out of focus light” to confound the signal from single cells, as has been the case for

implanted CIS that rely on contact imaging. This ability to correctly assign the detected light to the source it came from in three dimensions is shared with confocal and two-photon imaging, and is one of the reasons they are such powerful techniques. In those cases, the hardware design ensures nearly all detected fluorescent light originates from within the focal spot; in the case of the ILM, the software algorithm determines the distribution of fluorescence from the full 3-D constellation of sources each frame, based on the acquired light field data. To achieve this, we sacrifice diffraction-limit resolution and use post-hoc knowledge, two tradeoffs not required by confocal and multiphoton imaging.

Shot noise is particularly impactful in an ILM because of our 3-D single-shot imaging strategy. The shot noise on a pixel is proportional to the square root of the total number of photons—from all sources—hitting the pixel, while the signal from a given source on that pixel is of course only proportional to the number of photons hitting the pixel from that one source. This poses little problem for nearby sources, as they contribute a substantial fraction of the total number of photons hitting the pixel. For more distant neurons, however, this is the primary contributor to reconstruction error: The signal photon count on a single pixel from a distant source is smaller than the shot noise level. As noted in II.B.1, this problem can be mitigated by generating greater photon counts: brighter fluorophores, increased excitation intensity, longer exposure time, etc. It could also be substantially mitigated using any of a number of structured illumination techniques: When the nearby sources are not illuminated with excitation light, they will not contribute the large quantity of shot noise that impairs reconstruction of distant sources.

An interesting aspect of our lensless technique is that any given pixel is likely to have much greater shot noise than signal from any particular source—even a nearby source. The reason an ILM can achieve good signal-to-noise ratios is that the signal from any source is spread across a large number of pixels—unlike in focused imagers, where the light from a localized source is focused onto one or a few pixels—and the signal on each pixel is combined in the linear algebra process outlined in Figure 18. The sum of low SNR signals on many pixels can thus yield a high SNR result for the source.

Because the imager is near the target to be imaged, we collect a very large fraction of the light emitted from a source. Our simulated imager would collect light from over 2.6 steradians from a source centered 50 μm away. Half of the light is blocked by the patterned mask, but the effective numerical aperture from a light collecting perspective is still over 1.3. Nearer sources would have an even higher effective NA; off-center or more distant sources would have a lower effective NA, a fact that is manifest in the strong variation of the impact of shot noise with distance noted above.

The framework of using a light transport matrix that maps light from voxels to sensor images is particularly useful in that it is generalizable to other types of priors and reconstruction algorithms. If post-hoc knowledge is chosen as the prior of interest, rather than calculating an \mathbf{A} matrix from all voxels, the image on the sensor from each source can be calculated, directly generating the matrix \mathbf{N} . Positivity of the fluorescence power from each cell—and nonnegative matrix factorization⁹⁹ techniques, or sparsity of the distribution of neuronal nuclei—and L1 minimization^{100,101} techniques—are other examples of additional knowledge of the system that can constrain the solution.

While we briefly addressed the impact of errors in the spatial information priors, we did not explore techniques for correction of such errors, such as simulated annealing over changes to the matrix \mathbf{N} or applying parametrized coordinate transformations to the spatial priors (and thus generating new

families of matrices N'), in both cases seeking to find the N' that minimizes the error between the sensor image and the image predicted by corresponding solution M' : $\|Y - N'M'\|_2$.

It may also be possible for the ILM to bootstrap its own spatial information priors. Using feature extraction on the data (such as principal- or independent- component analysis, as used in^{87,90}), combined with knowledge of the light transport matrix, might allow deduction of source locations. Multiphoton or structured illumination techniques that are robust in the presence of scattering (such as⁶²) could also potentially allow excitation of a single source even at relatively large depths, allowing the ILM to determine the sensor image from each source sequentially, before or after the physiological imaging experiment.

In the general case, the voxel size is a tradeoff: Smaller voxels can yield higher resolution reconstructions, but they also render the equation $y = A * x$ even more underconstrained. Small voxels also worsen the condition number of the matrix A —the smaller the voxel, the more similar two adjacent voxels' columns in A , increasing any errors when solving for x . 4 micron voxels are smaller than the simulated nuclei, which prevents two sources from lying entirely inside the same voxel and completely mixing their signals. At the same time, 4 micron voxels do not overwhelm a desktop PC (in memory size or computation time) or yield highly underconstrained equations.

As noted in II.B.2, passive cooling is sufficient to keep brain temperature change to +/-1 C, but for a given probe geometry only a certain range of frame rates will keep the temperature in that range: Lower frame rates lead to more brain cooling, while higher frame rates cause more heating. Adding a small Peltier element between the external electronics and a small (on the order of 5 mm²) heat sink would allow heat dissipation to be increased or decreased, keeping temperature changes in the brain to less than one degree over a much larger range of frame rates, including when the imager is not in use, such as between experimental trials.

E. Future directions

1. Building an ILM

The most straightforward next step would be to build the probe and use it experimentally. Upfront costs for this probe are relatively high: A custom image sensor chip is required and would need to first be designed. The lithographic masks would need to be made. A run of ILM wafer(s) would need to be performed in a relatively advanced fabrication facility that can make back side illuminated, small pixel CMOS imagers. Next would come careful post processing (adding of excitation filter, mask deposition and etching, passivation/biocompatibility layers, etc.) of the silicon chips, as well as cutting the wafer into the individual probe shanks or arrays of probe "combs". The final step before the ILM would be implantation-ready is the off-chip connections, electrical, mechanical, and thermal. This would be wire attachment, addition of any external cooling, and mechanical attachment to any headplate or jointly implanted device (such as an excitation lightguide probe).

The off-chip connections all have close analogs to work our lab has already done for optogenetics; although the work is sensitive, it is a well-solved problem. See in particular^{30,52,53} among others.

The post-processing of the silicon dies also requires little out of the ordinary; I attempted to perform these post-processing steps on off-the-shelf sensors, and the only thing preventing this was difficulty acquiring bare image sensor dies: all commercially available options at less-than-wafer counts come

already encapsulated and with lenslets and IR filters already applied. Removing the lenslets while keeping the wafer clean enough to enter facilities that could then do the post-processing was more trouble than it was worth for this early stage of the project, but might be merited now.

The design and fabrication of the ILM's CMOS image sensor is also not terribly difficult, but it is expensive. As demonstrated above, existing pixel designs are sufficient, so the chip design involves only licensing one of those pixel designs and laying out the pixels in the unusual aspect ratio of an ILM. Note that existing chips often already have all of the line drivers, ADCs, amplifiers, etc. set off to one side of the imaging pixels, and the sensor width is typically several millimeters. The ILM's requirements that active electronics be located at one end—outside the brain—as much as possible, and probe shanks of several millimeters are thus already standard practice. Making the lithographic masks and interrupting an advanced fab's normal workflow are expensive propositions; it is an example of "the first one costs a million dollars, and all the rest cost ten." This more than anything else is what necessitated such thorough simulation to determine whether the ILM was worth making. The findings presented above demonstrate that the high upfront cost of designing and building the actual probe are very likely to be worth it. Commercially-driven cost decreases also rapidly lower the cost bar over time, which is why I do not quote specific numbers here. As more advanced processes inevitably make their way to smaller custom fab houses, the difficulty of actually getting a wafer run allocated and its lead time drop precipitously, making ILM creation easier and faster.

2. A first ILM experiment

Perhaps the best first experiment would be a joint ILM plus multiphoton imaging of activity in a deep mouse brain structure. Not only could precise two photon excitation of target neurons provide anatomical priors for the ILM, but the ILM could overcome some of the limits of two photon microscopy. Additionally, two photon imaging in mice *in vivo* is already well-established, and ILMs could be added without overly much disruption to the existing workflow. The easiest would be a headfixed imaging experiment, where the ILM would be implanted when the headplate was attached, then attached to the two photon's stage at a later date for an awake, headfixed experiment. More technically difficult—but still viable and even more experimentally powerful—would be using a head-mounted two photon microscope (as in⁸¹). The existing two photon apparatus could have an ILM attached to it, and the two photon positioned at the brain surface while the ILM penetrated. This setup would allow freely moving behavior, but would be less forgiving about positioning and hardware function, as the two elements are permanently affixed to the mouse.

Multiphoton microscopy is fundamentally depth limited by two mechanisms: Due to absorption and scattering, at a depth of 1-2mm, the excitation focal spot's intensity is less than the intensity at the surface of the brain—even though the excitation light at the surface is not focused—and the detected emission photons are dominated by those emitted at the surface rather than the desired focal spot. This can be mitigated by using the highest possible NA lenses (for greatest surface-to-focus intensity ratio) and avoiding fluorophores near the surface of the brain when possible, but is often the dominant limiting factor in real experiments. A second limitation is the fact that the emission light must be detected, normally by traversing all of the intervening brain tissue back to the brain surface. Adding to the problem, this emitted light is back in the visible range and doesn't benefit from the (typically IR) excitation light's improved penetration depth. This second issue essentially exacerbates the problem noted in the first.

The ILM would substantially circumvent both of these multiphoton difficulties: For the first, by using its ability to detect position in 3D, the ILM can distinguish between emission light from the focal spot and from near the surface. As noted, when the multiphoton imaging breaks down, these two locations are separated by at least a millimeter and don't require precise anatomical knowledge to distinguish. Using knowledge of where the multiphoton is focused, we also generate our anatomical priors. More precisely, we generate actual in vivo sensor images of the source—thus avoiding possible problems in numerical simulations that don't account for diffraction, mask defects, etc. The ILM, by being implanted adjacent to the target brain region, is also much closer to the source of the visible range emission light. Rather than traveling over a millimeter to the surface, it travels perhaps 50 microns to the implanted ILM. And, as noted, the ILM has a high effective numerical aperture from a light collecting perspective. These are rather compelling synergies for a first use of an ILM. The limiting factors on depth for such an ILM-plus-multiphoton experiment would become practical ones, such as available excitation power and phototoxicity. Eventually, another fundamental depth limit would be encountered: Scattering (even from IR excitation light) would become severe enough that the number of nearly ballistic photons that reach the nominal focal spot would be small compared to the total number of photons at that depth, and the focal spot would be spread to a size too large to identify single neurons. Even here, an ILM that acquired anatomical knowledge—from, for instance, a post-hoc slicing of the brain—would be able to use even the spread focal spot as excitation light and observe the neural activity. In this regime the excitation light would need to be incredibly intense.

3. Excitation sources and their use in improving ILM performance

This thesis and the submitted paper on the ILM consciously ignores the excitation illumination question. As noted, it is something of a solved problem, given that excitation illumination for optogenetics has been an important topic for years and has seen a number of viable solutions. However, specific solutions are in some circumstances technically “tricky”—such as a combined waveguide/ILM probe requiring very tight tolerances and careful micromachining. Other solutions, such as an ILM attached to- and extending beyond- an excitation fiber would be quite straightforward, as mouse neural implants go. One such configuration could put a large fiber flush with the surface of the brain and record from all layers of the cortex with the ILM. Another design would be a microscope objective delivering the excitation light from outside the brain while the ILM records emission, as in the ILM-multiphoton experiment described above.

More rewarding might be work on ways to incorporate structured illumination with the ILM. The simplest design would be simply two sources, one illuminating cells near the ILM and one cells farther away. By alternating which source is on, you would sample the nearby cells with one frame, then the farther cells with the next—and without the highly problematic shot noise from the nearby neurons. A scanning light sheet would likewise help with the problem. Clearly, there is a frame rate tradeoff here, and we would be blunting one of the ILM's advantages over scanning modalities, but I suspect some tradeoff might be highly beneficial.

More complicated structured illumination could also be very fruitful. As noted in II.A.2, each pixel in an ILM samples from a random projection of the light field (as determined by the quasirandom mask). Because there is a direct mapping between the light field and the 3D space, this is equivalent to saying that each pixel samples from a random projection of the 3D space. One could create an “inverse ILM”, where one replaces the imaging pixels with LED emitters that could illuminate random projections of the

3D space—though a practical realization would probably not literally use LED emitters where the pixels are, as LEDs are generally not as miniaturized at the required powers, and generate additional heat inside the brain. Such an excitation device could make the resulting linear algebra problem much better conditioned.

A general way to think of excitation light is as follows: The form $Y = \mathbf{A}X$ (as defined in II.A.2) presents X as the fluorescence from a voxel; that physical quantity can be broken down into a product of two factors, the excitation light fluence, and the level of the value measured by the fluorescent reporter (e.g., calcium concentration for GCaMPs) in that voxel. The latter quantity is the quantity to determine, and the former is (via structured illumination) controlled by the experimenter. Because the two quantities interact linearly (at least in the typical in vivo regime where the fluorophore is not near excitation saturation nor calcium saturation), we can re-express the linear algebra form as

$$Y = \mathbf{A}(\mathcal{J} \circ \mathcal{P})$$

Where \mathcal{J} is the excitation fluence at each voxel, \mathcal{P} is the physiological activity level as reported by the fluorescent molecule, and \circ is the Hadamard product. \mathcal{J} is in turn given by a light transport matrix equation—this time from the excitation light sources to the voxels, somewhat analogous to an inverse of the \mathbf{A} matrix of emission light to the ILM’s sensor:

$$\mathcal{J} = \mathbf{B}^{-1}L$$

Where I have defined the equation using \mathbf{B}^{-1} rather than \mathbf{B} to keep the same sense of direction of propagation for both light transport matrices; no actual inverse need be calculated, and \mathbf{B}^{-1} can be estimated via Monte Carlo solutions (this is in fact a way to represent the work in part I of this thesis). As we have some control over \mathbf{A} by designing the mask pattern, so do we have some control over \mathbf{B} by the design of our structured illumination. This is also a forward equation: we have full control over L , the intensity of each source in our structured illumination setup. The resulting equation becomes

$$Y = \mathbf{A}([\mathbf{B}^{-1}L] \circ \mathcal{P})$$

or, eliminating the Hadamard product by introducing a diagonal matrix

$$Y = \mathbf{A} \text{diag}(\mathbf{B}^{-1}L)\mathcal{P}$$

While these equations do not quickly simplify in the general case—especially with physical constraints imposed on the light transport matrices—it is clear that structured illumination adds independent, controllable variables that can aid in the solving for the underlying activity \mathcal{P} . It is also suggestive of some directions for optimizing those variables. For example, the matrix $\mathbf{A} \text{diag}(\mathbf{B}^{-1}L)$ should be as well-conditioned as possible.

4. Improved mathematical frameworks

The other major area ripe for further research are the mathematical techniques surrounding solving the underconstrained linear algebra problem. Unsurprisingly, this is in turn intimately related to ILM and illumination source design.

While there are generally many more voxels than pixels, there are far fewer neurons than voxels—and typically fewer neurons than pixels, rendering the system potentially overconstrained, though see

below. Sparsity in the object space lends itself naturally to L_1 minimization techniques such as basis pursuit denoising (BPDN), in which the following equation is solved to generate a solution:

$$\operatorname{argmin}_x (|y - Ax|_2^2 + \lambda|x|_1)$$

Where the added L_1 term “punishes” small components of x more than a pure least-squares solution, pushing the result towards a sparse x . The cell body volume of brain tissue is on the order of 5%, which is borderline for “sparse enough”, and all theorems proving that L_1 techniques yield correct solutions require the matrix A to obey the restricted isometry property (RIP), which the light transport matrices do not. This is not an if and only if relation, though; L_1 techniques may still yield good solutions in the absence of a strong RIP value.

Early attempts at using L_1 techniques to constrain the solution in lieu of anatomical priors did not work as a standalone solution. I went on to try to decompose the sensor images using ICA, stICA, PCA, and related techniques over the full (simulated) sensor videos. The motivating idea is that if ICA could decompose the sensor images into components where each component represented one (or even several; just not hundreds) neuron, then each component would be extremely sparse and could be separately solved via L_1 methods, even with RIP limitations. These attempts were promising, but did not succeed. Some components would separate well, but some neurons’ sensor images would be split across components, and separating useful components with modest SNR from noise components was subtle. Still, with some stitching together of components and more careful treatment of the analysis, this could prove powerful and could allow the ILM to generate its own anatomical knowledge rather than needing outside information.

The linear algebra problem as formulated operates on each frame independently, which ignores the temporal properties of the signals to be recorded: Fluorescence from the neurons and the resulting sensor images are not uncorrelated between frames. Also, if the physiological activity to be recorded is neural spiking, the signal is sparse in time as well as space. Introduction of ICA, etc. represent the first step in including time information (by noting correlated variations between different frames), but is by no means the only method that can be used. Spike traces will have a characteristic fluorescence time course that can be used with template matching—especially given that the time course is (usually) determined by the rise and fall times of the fluorescent molecule and known in advance. If the neurons occupy 5% of the volume and spiking is at most 10% of the time (which would correspond to a 1ms spike duration firing at 100 Hz, which is possible but certainly far above average firing rates), the 4D recorded volume over time is 0.5%. This much improved sparsity would be a boon for L_1 techniques.

As noted, the light transport matrix does not obey the RIP. Relatedly, its condition number is poor when it includes voxels at large distances from the sensors. This is because at large distances, two nearby voxels generate nearly the same sensor images, due to the parallax between them being small. This results in nearly identical columns of the light transport matrix. This problem is probably the least tractable. For a given mask-sensor distance, no optimization of the mask will improve this very much; only smaller pixels that can detect the smaller parallaxes. One thing that does help is masks at a greater distance: the mask pinhole plus the two sources and the mask pinhole plus neighboring pixels form similar triangles, so moving the mask farther from the sensor plane increases the parallax distance to be detected at the sensor. The downsides of this are twofold: having the mask farther from the sensor allows light from sources at nearer distances passing through a particular pinhole to spread farther

across more pixels of the sensor and renders the sensor images as similar blobs rather than crisp patterns. It also makes the device thicker. A hybrid solution of two (or more) masks at varying distances, with a total opacity of ~50% might “split the difference” in a useful way, but early simulations didn’t show great benefit. Careful separation of this issue from the shot noise issue is required.

5. Imager improvements

Given the dominant limitations being due to far neurons (both shot noise and small parallax issues), optimizing the mask pattern did not have an appreciable impact on performance. I used a quasi-random mask that avoided very large open or opaque areas, and was not periodic over short scales—thus avoiding aliasing issues. Other patterns more intelligently designed, such as MURA codes performed indistinguishably. However, this is not to say that this would be true for any method of solving the linear algebra formulation. Techniques such as BPDN or more advanced strategies like the use of overcomplete dictionaries¹⁰¹ might benefit from more careful mask design. A dynamic mask with no moving parts—such as electrochromic materials or thin LCD masks—could also prove useful, especially if it could be varied at nearly the imager frame rate.

This work uses only conventional (in pixel design if not form factor) CMOS image sensors, but ILMs could benefit from more unusual imagers. The angle sensitive pixels of, e.g.¹⁰² are interesting, but their need to make a diffraction grating for each (set of) angular sensitivities make their subpixels too large, and in a way dictated by the wavelength of light. A related idea that might work better would be to intentionally make a complicated diffraction pattern by using a mask with very small feature sizes—and for this purpose the mask could be a phase mask that did not reject half the light, as the mask described here does. The diffraction pattern would again depend on the position and angle of the incident photon and would thus include the needed spatiotemporal information. By not requiring a pixel to collect a specific, contiguous volume of the light field, such a device could overcome the insufficient spatial resolution of the pixels described in¹⁰², just as this thesis overcame the insufficient spatial resolution provided by widely spaced pinholes.

The pixels as described sample from a random subspace of the light field due to their pinhole mask. This concept could be extended to the time domain by triggering the reset and readout on the pixels at quasirandom times. Because a given source’s signal is spread over many pixels in an ILM, sampling subsets at various times could—again, by leveraging computational imaging—allow for improved temporal resolution, potentially much finer than the integration time of any given pixel. While per-pixel randomized triggering would not be easy without additional active electronics at the pixel (and consequently in the brain), per-column randomized triggering would be very easy, likely requiring only software changes, or at worst, hardware changes outside the brain.

Generally speaking, any additional information about the incoming light (time of flight, polarization, exact frequency, etc.) can possibly be imprinted with useful information for activity recording and/or signal separation. The difficulty for any of these techniques is often either spatial resolution or photon gathering efficiency (e.g., a SPAD can measure time of flight and detect single photons, but given its refractory period, it cannot detect them at a high enough rate to overcome shot noise issues, and is too large for the required spatial resolution). Also, fluorescent reporters generate signals with relative broad spectra, distributed over a relatively long decay time (over time-of-flight time scales), nearly unpolarized, and incoherent. All of these features of fluorescence introduce difficulties in using more

sophisticated information about the incoming photons. Signals must also be resolved in a single trial for most neuroscience purposes, as well, and many advanced optical techniques require (for, e.g. limiting shot noise, or to image the entire volume/time series).

6. Brain temperature changes induced by implants and craniotomies

Temperature changes of the brain around craniotomies and thermally conductive implants is drastically underappreciated. I directly observed this in another unpublished project, where the cortex of a mouse with a saline-irrigated craniotomy was measured with a needle thermocouple and observed to fall to within 2C of room temperature. I also observed this in the implant simulations in II.B.2, and it is consonant with the physical intuition that extremely high thermal conductivities introduced into the brain will change its temperature. The impact of temperature on brain function is profound; the temperature variations I observed can cause spiking to vary by an order of magnitude, plus changes to nearly all reaction kinetics of cellular biochemistry.

A simple experiment could be straightforward yet powerful and impactful, improving the quality of results across a huge range of neuroscience experiments. The experiment could consist of showing craniotomy temperature using a thermal camera for part one. Part two could be an experiment using needle thermocouples acutely implanted (into an irrigated craniotomy) and chronically implanted (cemented in place with dental acrylic or other relevant implant attachment method) to measure temperature at depth in those circumstances. Needle thermocouples are commercially available in diameters of 75 microns and above and made primarily of metals, making them reasonable stand-ins for most kinds of implantable electrodes. The straightforward physics of thermocouples might render it possible to make custom thermocouples of even smaller size to mimic even smaller electrodes such as tetrodes. For silicon probes, a thermocouple could likely be deposited directly on the silicon itself with only modest fabrication difficulty.

Table 1

Properties of light scattering and absorption of gray matter at 532nm used in Monte Carlo simulation	
Absorption coefficient ³⁶	0.942 mm ⁻¹
Scattering coefficient ³⁶	23.3 mm ⁻¹
Anisotropy factor ³⁶	0.949
Phase function ³⁶	Gegenbauer kernel

Table 2

Properties related to GCaMP6f used in fluorescent activity simulation	
dF/F0 of single spike ⁶⁶	0.19
Rise time (baseline to peak) ⁶⁶	45ms
t _{1/2} Decay half-life ⁶⁶	142ms
fluorophores per cell ^{103,104}	10 ⁷
F0 in neurons (relative to EGFP) ^{66,92,93}	8.9%
EGFP extinction coefficient ¹⁰⁵	53,000 M ⁻¹ cm ⁻¹
EGFP Quantum efficiency ¹⁰⁵	0.6
Excitation fluence	31mW/mm ²
Energy of 465nm excitation photon	4.27*10 ⁻¹⁹ J

Table 3

Properties related to CMOS image sensor pixels used in imaging and heating simulations	
Pixel size ⁹⁴	1.12 μm
Total energy per pixel per frame ⁹⁴	0.5 nJ
Percent of total power generated at pixels ^{95,98}	16.7%
Bits per pixel ⁹⁴	10
RMS read noise ¹⁰⁶	2.2 e^-
Sensitivity ¹⁰⁶	4800 $e^-/\text{lx/s}$
Quantum efficiency ¹⁰⁶	0.6
Saturation Signal ¹⁰⁶	5000 e^-
Luminous efficacy at 532nm ¹⁰⁷	610 lm/W
Energy of 532nm emission photon	3.73×10^{-19} J
Thermal conductivity of silicon ⁹⁶	163 W/m/K
Thermal conductivity of absorption filter ⁹⁶	0.18 W/m/K
Thermal conductivity of brain tissue ⁹⁷	0.565 W/m/K
Density of silicon ⁹⁶	2330 kg/m ³
Density of absorption filter ⁹⁶	1190 kg/m ³
Density of brain tissue ⁹⁷	1,039 kg/m ³
Heat capacity of silicon ⁹⁶	703 J/kg/K
Heat capacity of absorption filter ⁹⁶	1470 J/kg/K
Heat capacity of brain tissue ⁹⁷	3,680 J/kg/K
Blood perfusion cooling coefficient ⁹⁷	35,000 W/m ³ /K
Brain metabolism ⁹⁷	10,000 W/m ³

1. L. Acker et al., "FEF inactivation with improved optogenetic methods.," Proc. Natl. Acad. Sci. U. S. A. **113**(46), E7297–E7306, National Academy of Sciences (2016) [doi:10.1073/pnas.1610784113].
2. A. M. Packer, B. Roska, and M. Häusser, "Targeting neurons and photons for optogenetics.," Nat. Neurosci. **16**(7), 805–815, Nature Publishing Group (2013) [doi:10.1038/nn.3427].
3. R. Pashaie et al., "Optogenetic Brain Interfaces," IEEE Rev. Biomed. Eng. **7**, 3–30 (2014) [doi:10.1109/RBME.2013.2294796].
4. M. MacDougall et al., "Optogenetic manipulation of neural circuits in awake marmosets," J. Neurophysiol. **116**(3) (2016).
5. K. L. Montgomery et al., "Beyond the brain: Optogenetic control in the spinal cord and peripheral nervous system," Sci. Transl. Med. **8**(337) (2016).
6. Y. Cho, C. L. Zhao, and H. Lu, "Trends in high-throughput and functional neuroimaging in *Caenorhabditis elegans*," Wiley Interdiscip. Rev. Syst. Biol. Med., e01376, John Wiley & Sons, Inc. (2017) [doi:10.1002/wsbm.1376].
7. E. J. Izquierdo and R. D. Beer, "The whole worm: brain–body–environment models of *C. elegans*," Curr. Opin. Neurobiol. **40**(40), 23–30 (2016) [doi:10.1016/j.conb.2016.06.005].
8. K. Appasani, Ed., *OPTOGENETICS: From Neuronal Function to Mapping & Disease Biology*, Cambridge University Press (2015).

9. C. Towne et al., "Overview on Research and Clinical Applications of Optogenetics," in *Current Protocols in Pharmacology*, p. 11.19.1-11.19.21, John Wiley & Sons, Inc., Hoboken, NJ, USA (2016) [doi:10.1002/cpph.13].
10. E. S. Boyden et al., "Millisecond-timescale, genetically targeted optical control of neural activity," *Nat. Neurosci.* **8**(9), 1263–1268, Nature Publishing Group (2005) [doi:10.1038/nn1525].
11. A. M. Aravanis et al., "An optical neural interface: in vivo control of rodent motor cortex with integrated fiberoptic and optogenetic technology.," *J. Neural Eng.* **4**(3), S143-56, IOP Publishing (2007) [doi:10.1088/1741-2560/4/3/S02].
12. O. Yizhar et al., "Optogenetics in neural systems.," *Neuron* **71**(1), 9–34 (2011) [doi:10.1016/j.neuron.2011.06.004].
13. K. Deisseroth, "Predicted irradiance values: model based on direct measurements in mammalian brain tissue," <<http://web.stanford.edu/group/dlab/cgi-bin/graph/chart.php>> (accessed 8 March 2016).
14. A. N. Yaroslavsky et al., "Influence of the Scattering Phase Function Approximation on the Optical Properties of Blood Determined from the Integrating Sphere Measurements," *J. Biomed. Opt.* **4**(1), 47–53, SPIE (1999) [doi:10.1117/1.429920].
15. A. N. Yaroslavsky et al., "Optical properties of selected native and coagulated human brain tissues in vitro in the visible and near infrared spectral range," *Phys. Med. Biol.*(12), 2059 (2002).
16. M. Cavaglia et al., "Regional variation in brain capillary density and vascular response to ischemia," *Brain Res.* **910**(1–2), 81–93 (2001) [doi:10.1016/S0006-8993(01)02637-3].
17. J. Sun et al., "Refractive index measurement of acute rat brain tissue slices using optical coherence tomography.," *Opt. Express* **20**(2), 1084–1095 (2012).
18. D. E. J. G. J. Dolmans, D. Fukumura, and R. K. Jain, "TIMELINE: Photodynamic therapy for cancer," *Nat. Rev. Cancer* **3**(5), 380–387, Nature Publishing Group (2003) [doi:10.1038/nrc1071].
19. W. M. Star, "Light dosimetry in vivo," *Phys. Med. Biol.* **42**(5), 763 (1997).
20. J. Marijnissen and W. Star, "Quantitative light dosimetry in vitro and in vivo," *Lasers Med. Sci.* **2**(4), 235–242, Springer London (1987) [doi:10.1007/bf02594166].
21. M. L. de Jode, "Monte Carlo simulations of the use of isotropic light dosimetry probes to monitor energy fluence in biological tissues," *Phys. Med. Biol.* **44**(12), 3027 (1999).
22. H. Van Staveren et al., "Construction, quality assurance and calibration of spherical isotropic fibre optic light diffusers," *Lasers Med. Sci.* **10**(2), 137–147, Springer London (1995) [doi:10.1007/bf02150852].
23. J. P. A. Marijnissen and W. M. Star, "Calibration of isotropic light dosimetry probes based on scattering bulbs in clear media," *Phys. Med. Biol.* **41**(7), 1191 (1996).
24. J. P. A. Marijnissen and W. M. Star, "Performance of isotropic light dosimetry probes based on scattering bulbs in turbid media," *Phys. Med. Biol.* **47**(12), 2049 (2002).
25. H. B. Henninger et al., "Validation of computational models in biomechanics," *Proc. Inst. Mech. Eng. Part H J. Eng. Med.* **224**(7), 801–812, SAGE Publications (2009) [doi:10.1243/09544119JEIM649].
26. R. B. Fernandez-Prini, Roberto; Dooley, "Release on the Refractive Index of Ordinary Water

Substance as a Function of Wavelength, Temperature and Pressure” (1997).

27. M. Jang et al., “Relation between speckle decorrelation and optical phase conjugation (OPC)-based turbidity suppression through dynamic scattering media: a study on in vivo mouse skin.,” *Biomed. Opt. Express* **6**(1), 72–85, Optical Society of America (2015) [doi:10.1364/BOE.6.000072].
28. J. D. Briers, “Laser Doppler, speckle and related techniques for blood perfusion mapping and imaging,” *Physiol. Meas.* **22**(4), R35–R66, IOP Publishing (2001) [doi:10.1088/0967-3334/22/4/201].
29. R. C. Mesquita et al., “Direct measurement of tissue blood flow and metabolism with diffuse optics,” *Philos. Trans. A. Math. Phys. Eng. Sci.* **369**(1955), 4390–4406 (2011) [doi:10.1098/rsta.2011.0232].
30. J. G. Bernstein et al., “Prosthetic systems for therapeutic optical activation and silencing of genetically targeted neurons,” *Proc. SPIE--the Int. Soc. Opt. Eng.* **6854**, 68540H–68540H–11, International Society for Optics and Photonics (2008) [doi:10.1117/12.768798].
31. J. W. Pickering et al., “Double-integrating-sphere system for measuring the optical properties of tissue.,” *Appl. Opt.* **32**(4), 399–410, Optical Society of America (1993) [doi:10.1364/AO.32.000399].
32. J. W. Pickering et al., “Two integrating spheres with an intervening scattering sample,” *J. Opt. Soc. Am. A* **9**(4), 621, Optical Society of America (1992) [doi:10.1364/JOSAA.9.000621].
33. I. V Yaroslavsky et al., “Inverse hybrid technique for determining the optical properties of turbid media from integrating-sphere measurements.,” *Appl. Opt.* **35**(34), 6797–6809, Optical Society of America (1996) [doi:10.1364/AO.35.006797].
34. B. Y. Chow et al., “High-performance genetically targetable optical neural silencing by light-driven proton pumps,” *Nature* **463**(7277), 98–102, Macmillan Publishers Limited. All rights reserved (2010) [doi:http://www.nature.com/nature/journal/v463/n7277/supinfo/nature08652_S1.html].
35. I. Kahn et al., “Characterization of the functional MRI response temporal linearity via optical control of neocortical pyramidal neurons,” *J. Neurosci.* **31**(42), 15086–15091 (2011).
36. A. S. Chuong et al., “Noninvasive optical inhibition with a red-shifted microbial rhodopsin,” *Nat Neurosci* **17**(8), 1123–1129, Nature Publishing Group, a division of Macmillan Publishers Limited. All Rights Reserved. (2014) [doi:10.1038/nn.3752 <http://www.nature.com/neuro/journal/v17/n8/abs/nn.3752.html#supplementary-information>].
37. D. R. Hochbaum et al., “All-optical electrophysiology in mammalian neurons using engineered microbial rhodopsins,” *Nat. Methods* **11**(8), 825–833, Nature Publishing Group, a division of Macmillan Publishers Limited. All Rights Reserved. (2014) [doi:10.1038/nmeth.3000].
38. P. A. Murphy et al., “Notch4 normalization reduces blood vessel size in arteriovenous malformations,” *Sci. Transl. Med.* **4**(117), 117ra8 (2012) [doi:10.1126/scitranslmed.3002670].
39. A. Roggan et al., “Optical Properties of Circulating Human Blood in the Wavelength Range 400–2500 nm,” *J. Biomed. Opt.* **4**(1), 36, SPIE (1999) [doi:10.1117/1.429919].
40. J. Hatazawa et al., “Regional cerebral blood flow, blood volume, oxygen extraction fraction, and oxygen utilization rate in normal volunteers measured by the autoradiographic technique and the single breath inhalation method,” *Ann. Nucl. Med.* **9**(1), 15–21, Springer Japan (1995).

41. W. K. Purves et al., *Life: The Science of Biology*, W. H. Freeman (2003).
42. A. N. Bashkatov et al., "Optical properties of human cranial bone in the spectral range from 800 to 2000 nm," in *Saratov Fall Meeting 2005: Optical Technologies in Biophysics and Medicine VII*, V. V. Tuchin, Ed., pp. 616310-616310–616311, International Society for Optics and Photonics (2006) [doi:10.1117/12.697305].
43. N. Ugryumova, S. J. Matcher, and D. P. Attenburrow, "Measurement of bone mineral density via light scattering," *Phys. Med. Biol.* **49**(3), 469–483, IOP Publishing (2004) [doi:10.1088/0031-9155/49/3/009].
44. L. Wang, S. L. Jacques, and L. Zheng, "MCML—Monte Carlo modeling of light transport in multi-layered tissues," *Comput. Methods Programs Biomed.* **47**(2), 131–146 (1995) [doi:http://dx.doi.org/10.1016/0169-2607(95)01640-F].
45. E. Alerstam, T. Svensson, and S. Andersson-Engels, "Parallel computing with graphics processing units for high-speed Monte Carlo simulation of photon migration.," *J. Biomed. Opt.* **13**(6), 60504, International Society for Optics and Photonics (2008) [doi:10.1117/1.3041496].
46. Q. Fang and D. A. Boas, "Monte Carlo simulation of photon migration in 3D turbid media accelerated by graphics processing units.," *Opt. Express* **17**(22), 20178–20190, Optical Society of America (2009) [doi:10.1364/OE.17.020178].
47. N. Ren et al., "GPU-based Monte Carlo simulation for light propagation in complex heterogeneous tissues.," *Opt. Express* **18**(7), 6811–6823, Optical Society of America (2010) [doi:10.1364/OE.18.006811].
48. "PCI Express," Wikipedia, <https://en.wikipedia.org/wiki/PCI_Express#PCI_Express_2.0> (accessed 1 January 2016).
49. NVidia, "GTX 560 specifications," <<http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-560/specifications>> (accessed 1 January 2016).
50. L. C. Acker, "Novel Tools for Trans-dural Modulation of Large Brain Volumes," 2012 HST conference (2012).
51. E. S. Boyden et al., "Processes for design, construction and utilisation of arrays of light-emitting diodes and light-emitting diode-coupled optical fibres for multi-site brain light delivery," *J. Eng.* (2015) [doi:10.1049/joe.2014.0304].
52. A. N. Zorzos, E. S. Boyden, and C. G. Fonstad, "Multiwaveguide implantable probe for light delivery to sets of distributed brain targets.," *Opt. Lett.* **35**(24), 4133–4135, Optical Society of America (2010) [doi:10.1364/OL.35.004133].
53. A. N. Zorzos et al., "Three-dimensional multiwaveguide probe array for light delivery to distributed brain circuits.," *Opt. Lett.* **37**(23), 4841–4843, Optical Society of America (2012) [doi:10.1364/OL.37.004841].
54. X. Han et al., "Millisecond-timescale optical control of neural dynamics in the nonhuman primate brain.," *Neuron* **62**(2), 191–198, Elsevier (2009) [doi:10.1016/j.neuron.2009.03.011].
55. S. Thompson, L. Masukawa, and D. Prince, "Temperature dependence of intrinsic membrane properties and synaptic potentials in hippocampal CA1 neurons in vitro," *J. Neurosci.* **5**(3) (1985).
56. J. G. Bernstein and E. S. Boyden, "Optogenetic tools for analyzing the neural circuits of behavior," *Trends Cogn. Sci.* **15**(12), 592–600 (2011) [doi:10.1016/j.tics.2011.10.003].

57. A. N. Zorzos, E. S. Boyden, and C. G. Fonstad, "Multiwaveguide implantable probe for light delivery to sets of distributed brain targets," *Opt. Lett.* **35**(24), 4133, Optical Society of America (2010) [doi:10.1364/OL.35.004133].
58. M. R. Krames, "Status and Future Prospects for Visible-Spectrum Light-Emitting Diodes," in *SID Symposium Digest of Technical Papers* **47**(1), pp. 39–41 (2016) [doi:10.1002/sdtp.10594].
59. Y. Narukawa et al., "White light emitting diodes with super-high luminous efficacy," *J. Phys. D. Appl. Phys.* **43**(35), 354002, IOP Publishing (2010) [doi:10.1088/0022-3727/43/35/354002].
60. S. Nakamura and M. R. Krames, "History of Gallium–Nitride–Based Light-Emitting Diodes for Illumination," *Proc. IEEE* **101**(10), 2211–2220 (2013) [doi:10.1109/JPROC.2013.2274929].
61. S. Bernet et al., "Lensless digital holography with diffuse illumination through a pseudo-random phase mask," *Opt. Express* **19**(25), 25113–25124, OSA (2011).
62. B. Judkewitz et al., "Speckle-scale focusing in the diffusive regime with time-reversal of variance-encoded light (TROVE)," *Nat. Photonics* **7**(4), 300–305 (2013) [doi:10.1038/nphoton.2013.31].
63. N. C. Klapoetke et al., "Independent optical excitation of distinct neural populations," *Nat Meth* **11**(3), 338–346, Nature Publishing Group, a division of Macmillan Publishers Limited. All Rights Reserved. (2014).
64. X. Han et al., "Informational Lesions: Optical Perturbation of Spike Timing and Neural Synchrony Via Microbial Opsin Gene Fusions," *Front. Mol. Neurosci.* **2**, 12, Frontiers (2009) [doi:10.3389/neuro.02.012.2009].
65. J. Akerboom et al., "Genetically encoded calcium indicators for multi-color neural activity imaging and combination with optogenetics," *Front. Mol. Neurosci.* **6**, 2, Frontiers (2013) [doi:10.3389/fnmol.2013.00002].
66. T.-W. Chen et al., "Ultrasensitive fluorescent proteins for imaging neuronal activity," *Nature* **499**(7458), 295–300, Nature Publishing Group, a division of Macmillan Publishers Limited. All Rights Reserved. (2013) [doi:10.1038/nature12354http://www.nature.com/nature/journal/v499/n7458/abs/nature12354.html#supplementary-information].
67. P. Theer and W. Denk, "On the fundamental imaging-depth limit in two-photon microscopy," *J. Opt. Soc. Am. A* **23**(12), 3139–3149, OSA (2006).
68. F. Helmchen and W. Denk, "Deep tissue two-photon microscopy," *Nat Meth* **2**(12), 932–940 (2005).
69. N. G. Horton et al., "In vivo three-photon microscopy of subcortical structures within an intact mouse brain," *Nat. Photonics* **7**(3), 205–209, Nature Publishing Group (2013) [doi:10.1038/nphoton.2012.336].
70. F. Helmchen, "Miniaturization of fluorescence microscopes using fibre optics," *Exp. Physiol.* **87**(6), 737–745 (2002).
71. J. K. Kim, J. W. Choi, and S. H. Yun, "350- μ m side-view optical probe for imaging the murine brain in vivo from the cortex to the hypothalamus," *J. Biomed. Opt.* **18**(5), 50502, International Society for Optics and Photonics (2013) [doi:10.1117/1.JBO.18.5.050502].
72. M. Murayama and M. E. Larkum, "In vivo dendritic calcium imaging with a fiberoptic periscope system," *Nat. Protoc.* **4**(10), 1551–1559, Nature Publishing Group (2009).

73. I. N. Papadopoulos et al., "High-resolution, lensless endoscope based on digital scanning through a multimode optical fiber.," *Biomed. Opt. Express* **4**(2), 260–270 (2013) [doi:10.1364/BOE.4.000260].
74. J. C. Jung et al., "In Vivo Mammalian Brain Imaging Using One- and Two-Photon Fluorescence Microendoscopy," *J. Neurophysiol.* **92**(5), 3121–3133 (2004) [doi:10.1152/jn.00234.2004].
75. T. Kobayashi et al., "Functional brain fluorescence plurimetry in rat by implantable concatenated CMOS imaging system.," *Biosens. Bioelectron.* **53**, 31–36 (2014) [doi:10.1016/j.bios.2013.09.033].
76. K. Sasagawa et al., "An implantable micro imaging device for molecular imaging in a brain of freely-moving mouse," in 2014 IEEE International Symposium on Bioelectronics and Bioinformatics (IEEE ISBB 2014), pp. 1–4, IEEE (2014) [doi:10.1109/ISBB.2014.6820892].
77. Y. Sunaga et al., "Implantable CMOS imaging device with absorption filters for green fluorescence imaging," in SPIE BiOS, H. Hirschberg et al., Eds., p. 89280L, International Society for Optics and Photonics (2014) [doi:10.1117/12.2038948].
78. Y. Sunaga et al., "An implantable green fluorescence imaging device using absorption filters with high excitation light rejection ratio," in 2014 IEEE Biomedical Circuits and Systems Conference (BioCAS) Proceedings, pp. 248–251, IEEE (2014) [doi:10.1109/BioCAS.2014.6981709].
79. D. C. Ng et al., "An implantable and fully integrated complementary metal–oxide semiconductor device for in vivo neural imaging and electrical interfacing with the mouse hippocampus," *Sensors Actuators A Phys.* **145–146**, 176–186 (2008) [doi:10.1016/j.sna.2007.11.020].
80. C.-T. Ko et al., "A Novel 3D Integration Scheme for Backside Illuminated CMOS Image Sensor Devices," *Device Mater. Reliab. IEEE Trans.* **14**(2), 715–720 (2014) [doi:10.1109/TDMR.2014.2311887].
81. F. Helmchen, W. Denk, and J. N. D. Kerr, "Miniaturization of two-photon microscopy for imaging in freely moving animals.," *Cold Spring Harb. Protoc.* **2013**(10), 904–913 (2013) [doi:10.1101/pdb.top078147].
82. K. K. Ghosh et al., "Miniaturized integration of a fluorescence microscope," *Nat Meth* **8**(10), 871–878, Nature Publishing Group, a division of Macmillan Publishers Limited. All Rights Reserved. (2011) [doi:http://www.nature.com/nmeth/journal/v8/n10/abs/nmeth.1694.html#supplementary-information].
83. J. Scholvin et al., "Close-Packed Silicon Microelectrodes for Scalable Spatially Oversampled Neural Recording," *IEEE Trans. Biomed. Eng.* **PP**(99), 1–1 (2015) [doi:10.1109/TBME.2015.2406113].
84. E. Adelson and J. Bergen, "The Plenoptic Function and the Elements of Early Vision," *Comput. Model. Vis. Process.*, 3–20 (1991).
85. M. Levoy and P. Hanrahan, "Light field rendering," in Proceedings of the 23rd annual conference on Computer graphics and interactive techniques - SIGGRAPH '96, pp. 31–42, ACM Press, New York, New York, USA (1996) [doi:10.1145/237170.237199].
86. M. Levoy et al., "Light field microscopy," in ACM SIGGRAPH 2006 Papers on - SIGGRAPH '06 **25**(3), p. 924, ACM Press, New York, New York, USA (2006) [doi:10.1145/1179352.1141976].
87. R. Prevedel et al., "Simultaneous whole-animal 3D imaging of neuronal activity using light-field microscopy.," *Nat. Methods* **11**(7), 727–730, Nature Publishing Group, a division of Macmillan

Publishers Limited. All Rights Reserved. (2014) [doi:10.1038/nmeth.2964].

88. N. Vladimirov et al., "Light-sheet functional imaging in fictively behaving zebrafish.," *Nat. Methods* **11**(9), 883–884, Nature Publishing Group, a division of Macmillan Publishers Limited. All Rights Reserved. (2014) [doi:10.1038/nmeth.3040].
89. A. Berényi et al., "Large-scale, high-density (up to 512 channels) recording of local circuits in behaving animals.," *J. Neurophysiol.* **111**(5), 1132–1149 (2014) [doi:10.1152/jn.00785.2013].
90. E. A. Mukamel, A. Nimmerjahn, and M. J. Schnitzer, "Automated analysis of cellular signals from large-scale calcium imaging data.," *Neuron* **63**(6), 747–760 (2009) [doi:10.1016/j.neuron.2009.08.009].
91. A. Schüz and G. Palm, "Density of neurons and synapses in the cerebral cortex of the mouse," *J. Comp. Neurol.* **286**(4), 442–455, Alan R. Liss, Inc. (1989) [doi:10.1002/cne.902860404].
92. L. Tian et al., "Imaging neural activity in worms, flies and mice with improved GCaMP calcium indicators," *Nat Meth* **6**(12), 875–881, Nature Publishing Group (2009) [doi:http://www.nature.com/nmeth/journal/v6/n12/supinfo/nmeth.1398_S1.html].
93. T. Mao et al., "Characterization and Subcellular Targeting of GCaMP-Type Genetically-Encoded Calcium Indicators," *PLoS One* **3**(3), e1796, Public Library of Science, San Francisco, USA (2008) [doi:10.1371/journal.pone.0001796].
94. I. OmniVision Technologies, "OmniVision OV13850 product page," <<http://www.ovt.com/products/sensor.php?id=176>> (accessed 9 February 2015).
95. Y. Chae et al., "A 2.1 M pixels, 120 frame/s CMOS image sensor with column-parallel Delta Sigma ADC architecture," *IEEE J. Solid-State Circuits* **46**(1), 236–247 (2011) [doi:10.1109/JSSC.2010.2085910].
96. Comsol AB, "Comsol Multiphysics 4.3B," 4.3B (2013).
97. G. Lazzi, "Thermal effects of bioimplants," *Eng. Med. Biol. Mag. IEEE* **24**(5), 75–81 (2005).
98. I. Cevik et al., "An ultra-low power CMOS image sensor with on-chip energy harvesting and power management capability.," *Sensors (Basel)*. **15**(3), 5531–5554, Multidisciplinary Digital Publishing Institute (2015) [doi:10.3390/s150305531].
99. D. D. Lee and H. S. Seung, "Learning the parts of objects by non-negative matrix factorization.," *Nature* **401**(6755), 788–791 (1999) [doi:10.1038/44565].
100. D. L. Donoho, "Compressed sensing," in *Information Theory*, *IEEE Transactions on* **52**(4), pp. 1289–1306 (2006) [doi:10.1109/TIT.2006.871582].
101. K. Marwah et al., "Compressive light field photography using overcomplete dictionaries and optimized projections," *ACM Trans. Graph.* **32**(4), 1, ACM (2013) [doi:10.1145/2461912.2461914].
102. M. Hirsch et al., "A switchable light field camera architecture with Angle Sensitive Pixels and dictionary-based sparse coding," in *2014 IEEE International Conference on Computational Photography (ICCP)*, pp. 1–10, IEEE (2014) [doi:10.1109/ICCPHOT.2014.6831813].
103. H. Dana et al., "Thy1-GCaMP6 transgenic mice for neuronal population imaging in vivo," *PLoS One* **9**(9), 2014/09/25, e108697, Public Library of Science (2014) [doi:10.1371/journal.pone.0108697].

104. D. Huber et al., "Multiple dynamic representations in the motor cortex during sensorimotor learning.," *Nature* **484**(7395), 473–478, Nature Publishing Group, a division of Macmillan Publishers Limited. All Rights Reserved. (2012) [doi:10.1038/nature11039].
105. G. H. Patterson et al., "Use of the green fluorescent protein and its mutants in quantitative fluorescence microscopy.," *Biophys. J.* **73**(5), 2782–2790 (1997) [doi:10.1016/S0006-3495(97)78307-3].
106. N. Sukegawa, S.; Umebayashi, T.; Nakajima, T.; Kawanobe, H.; Koseki, K.; Hirota, I.; Haruta, T.; Kasai, M.; Fukumoto, K.; Wakano, T.; Inoue, K.; Takahashi, H.; Nagano, T.; Nitta, Y.; Hirayama, T.; Fukushima et al., "A 1/4-inch 8Mpixel back-illuminated stacked CMOS image sensor," in 2013 IEEE International Solid-State Circuits Conference Digest of Technical Papers, pp. 484–485, IEEE (2013) [doi:10.1109/ISSCC.2013.6487825].
107. "CIE - INTERNATIONAL COMMISSION ON ILLUMINATION: Selected Colorimetric Tables," <http://www.cie.co.at/index.php/LEFTMENU/index.php?i_ca_id=298> (accessed 26 June 2015).

III. Appendices

A. Photon propagation Monte Carlo code

next_photon_input_struct.m

```
%generates a data structure s that is passed to the function nextphoton,  
%which does all the work (the function form runs faster than a script with  
%this initialization at the beginning)  
  
%IF YOU ARE NOT SIMULATING A LIGHT DETECTING FIBER, just set the receive  
%tip position to be outside the simulated volume (or its diameter to zero)  
%and it will have no effect.  
  
X=1;  
Y=2;  
Z=3;  
  
millionsofphotons=2; %number of monte carlo photon packets to simulate, in  
millions  
s.nphotons=single(millionsofphotons*1e6);  
s.sendTipPos=single([0,2.5,2.5]); %the center of the tip of the fiber  
emitting excitation light  
s.sendTipDiam=single(.2); %diameter of the excitation fiber  
s.sendTipNorm=single([1,0,0]); %direction the excitation fiber is facing (ie,  
vector normal to the end face of the fiber)  
s.sendTipAA=single(21.7*pi/180); %half opening angle of emission (called AA  
to match with acceptance angle terminology of receive fiber), expressed in  
radians  
s.receiveTipPos=single([20,2,2]); %the center of the tip of the fiber  
receiving light (if any; just set this position to be outside simulated  
volume if none)  
s.receiveTipDiam=single(.2); %diameter of detecting fiber  
s.receiveTipNorm=single([0,1,0]); %direction of detection fiber (normal to  
end face)  
s.receiveTipAA=single(26*pi/180); %acceptance angle (in radians) of the  
detection fiber  
% size of tissue volume to be simulated (mm) need not be cubic  
s.blocksize=zeros(1,3,'single');  
s.blocksize(X)=single(5);  
s.blocksize(Y)=single(5);  
s.blocksize(Z)=single(5);  
%size of each voxel (mm). need not be cubic  
s.vox=zeros(1,3);  
s.vox(X)=single(.05);  
s.vox(Y)=single(.05);  
s.vox(Z)=single(.05);  
%number of voxels in each dimension  
s.nvox(X)=round(s.blocksize(X)/s.vox(X));  
s.nvox(Y)=round(s.blocksize(Y)/s.vox(Y));  
s.nvox(Z)=round(s.blocksize(Z)/s.vox(Z));
```

```

s.sidesrefl=false; %do the boundaries of the simulated volume along the Y and
Z axes reflect photons or absorb them? if true, they reflect; reflecting
boundary conditions are useful for simulating configurations with
translational symmetry in the YZ plane
%blocksize=s.blocksize;
%vox=s.vox;

```

volume_creator.m

```

function [absorb, scatter, g, tissueParam,tissueTypeEnum]=volume_creator(s)
%%% use this file to generated tissue volumes with parameters. First run
%%% next_photon_input_struct to load up some required variables, then write
%%% a script in this file to create the tissue properties you want, which
%%% are saved to disk. Load g, absorb, and scatter, then pass them to
%%% nextphoton.m.

```

```

%%% as far as choosing voxel size: It does NOT affect accuracy to change
%%% voxel size. The code can scatter a photon multiple times within
%%% a voxel if voxels are large, preserving accuracy. The voxel size
%%% determines the resolution of two things: 1) the resolution with which
%%% you can specify your tissue parameters (ie, if you have MRI data with
%%% XX resolution, you probably want to set your voxel size to XX) and 2)
%%% the resolution with which you record the fluence (isotropic light
%%% flux)... (ie, if you want to see how the excitation light varies across
%%% a brain structure of size YY, set voxel size to be substantially
%%% smaller than YY.) Small voxels increase computation time because the
%%% computation has to calculate every time the photon scatters OR hits a
%%% voxel boundary.

```

```

%%% the filename you will save to
save_tissue_name='593_grey.mat';

```

```

tissueTypeEnum={ % enum for tissue type. to get grey matter parameters for
465nm light, reference tissueParam(grey465,:)

```

```

'GREY465',int32(1); ...
'BLOOD465',int32(2); ...
'BLOOD633',int32(3); ...
'GREY633',int32(4); ...
'HCT1P8GREY465',int32(5); ...
'HCT1P8GREY532',int32(6); ...
'HCT1P8GREY593',int32(7); ...
'HCT1P8GREY633',int32(8); ...
'HCT1P8WHITE633',int32(9); ...
'NOSCAT',int32(10); ...
'WHITE633',int32(11); ...

```

```

'BONE633',int32(12); ... %values derived from a horse leg (Nadya
Ugryumova, Stephen John Matcher and Don P Attenburrow 2004). this squares
pretty well with bashkatov, tuchin, et al which cites ugryomova.

```

```

'WHITE532',int32(13); ...

```

```

'BONE532',int32(14); ...
'GREY532',int32(15); ...
'HCT1P8WHITE532',int32(16); ...
'BONE593',int32(17));
for ii=1:size(tissueTypeEnum,1) %above is rolled into a cell variable to
save it compactly; unroll it into an actual enum here
    %assign is not a matlab builtin; it's a one line function
    %bandaid for limitations in assignin
    assignin(tissueTypeEnum{ii,1},tissueTypeEnum{ii,2})
end
X=1;
Y=2;
Z=3;
ABS_COEF=int32(1);
SCAT_COEF=int32(2);
ANISO_COEF=int32(3);
PHASE_FUNCTION=int32(4);
HENYEYGREENSTEIN=single(1);
GEGENBAUERKERNEL=single(2); %these are different ways to calculate the
phase function and anisotropy factor, relevant for different tissues

%note human hematocrit levels in blood are typically 45% for males, and
%brain is 4% blood by volume
%the four values in each row of tissueparam are [mu_a,mu_s,g,(index
indicating phase function)]
%note for blood: it's VERY hard to get good values for blood at full
%45% hematocrit concentration, so the pure blood values below are
%extrapolations
%thankfully, outside of blood vessels, the brain "looks like"
%blood dilluted to 1.8% hematocrit (.45*.04). To get the values for
%1.8% hct, I extrapolate from the roggan and muller 1999 paper's values
%for 5% HCT at realistic osmolarity and flow rate (not that I trust
%flow rate over these scales, but its effects aren't large)
% the values for 5%hct (ie, dillute blood measured by roggan and muller)
are
%at 465nm: [3.2,41,.995]
%at 532nm: [2.5, 37,.993]
%at 593nm: [.6, 33, .991]
%at 633nm: [.15, 30, .991]

tissueParam(GREY465,:)=single([.07,11,.88,HENYEYGREENSTEIN]); %dynamic
reallocation is not a big deal for these small matrices
tissueParam(BLOOD465,:)=single([10,65,.95,GEGENBAUERKERNEL]); %blood is
for standard hematocrit levels (this is for 100% blood like inside a vessel)
tissueParam(GREY633,:)=single([.017,9.2,.90,HENYEYGREENSTEIN]); %if it
doesn't mention hct1p8 it doesn't have blood added)
tissueParam(BLOOD633,:)=single([.7,80,.97,GEGENBAUERKERNEL]);

```

```

tissueParam(HCT1P8GREY465,:)=single([1.22,25.8,.946,GEGENBAUERKERNEL]);
%hct1p8=1.8% hematocrit (which is normal blood level in brain) added
tissueParam(HCT1P8GREY532,:)=single([.942,23.3,.949,GEGENBAUERKERNEL]);
tissueParam(HCT1P8GREY593,:)=single([.246,21.4,.948,GEGENBAUERKERNEL]);
tissueParam(HCT1P8GREY633,:)=single([.071,20,.950,GEGENBAUERKERNEL]);
tissueParam(HCT1P8WHITE633,:)=single([.134,53.8,.87,GEGENBAUERKERNEL]);
tissueParam(HCT1P8WHITE532,:)=single([1,55.32,.862,GEGENBAUERKERNEL]);
tissueParam(NOSCAT,:)=single([0,0.00001,.99,GEGENBAUERKERNEL]); % no
scattering or absorption. scattering coefficient of 0.00001 to avoid div by
zero special casing, but this is a scattering length of 100 meters and a
reduced scattering length of 10km, so practically 0
tissueParam(WHITE633,:)=single([.08,43,.84,HENYEYGREENSTEIN]);
tissueParam(BONE633,:)=single([.06 ,33,.93,HENYEYGREENSTEIN]);
tissueParam(WHITE532,:)=single([.1,42,.82,HENYEYGREENSTEIN]);
tissueParam(BONE532,:)=single([.105,34,.93,HENYEYGREENSTEIN]);
tissueParam(GREY532,:)=single([.043,9.9,.89,HENYEYGREENSTEIN]);
tissueParam(BONE593,:)=single([.7,34,.93,HENYEYGREENSTEIN]);

%% generate shapes of different tissue structures and assemble the volume
to be simulated using the relevant tissue properties defined above

%this is a trivial case of the entire volume is uniform grey matter. Can
%write code to set the tissue properties per voxel (eg, by importing data
%from an MRI scan, modeling a toy skull/grey matter/white matter system,
%etc.)
greytype=GREY532;

absorb=ones(s.nvox(X),s.nvox(Y),s.nvox(Z),'single')*tissueParam(greytype,ABS_
COEF);

scatter=ones(s.nvox(X),s.nvox(Y),s.nvox(Z),'single')*tissueParam(greytype,SCA
T_COEF);
g=ones(s.nvox(X),s.nvox(Y),s.nvox(Z),'int32')*greytype; %this is NOT the
value of the parameter g, but rather the material index of the right theta
distribution. see calculation of randthetas in nextphoton.m

%% save the created tissue volume

save(save_tissue_name,'absorb','scatter','g','tissueParam','tissueTypeEnum')
end

```

nextphoton.m

```

function [outFlux,outPhotonPhase,outVox,outNVox,outBlockSize,nphotons] =
nextphoton(s,absorb,scatter,g,tissueParam)
%UNTITLED Summary of this function goes here
% by Michael Henninger originally 11-21-2010, based on earlier code photon
and photon2.

```

```

% last revision 2-12-2016 (code cleanup, comment improvements for public
release)
% lengths in mm unless specified
% variables are set to single (or int) precision because GPUs are (other
than some Teslas and Quadros) *much* faster
% with them, and none of the calculations on the GPU require double
precision.
    %rand('state',6515184) %for debugging use...if you set the rand seed
    %to a fixed value, you will get the same sequence of random numbers
    %every time you run the code, and all other settings equal, the same
    %photon trajectories. Change any parameters (scattering length, # of
    %photons, etc.) and that promise is void (for all photons).
    nphotons=s.nphotons;

%%%enumerations
    ABS_COEF=int32(1);
    SCAT_COEF=int32(2);
    ANISO_COEF=int32(3);
    PHASE_FUNCTION=int32(4);
    %%%NOTE that these constants are one-based in matlab but ZERO based in
the cuda code
    X=int32(1); %x position
    Y=int32(2); %y position
    Z=int32(3); %z position
    VX=int32(4); %x velocity component (velocity normalized to 1)
    VY=int32(5); %y velocity
    VZ=int32(6); %z velocity
    WEIGHT=int32(7); %relative weight of the simulated MC light "packet".
starts at one, decreases as the packet is partially absorbed while traveling
through tissue
    %these are different ways to calculate the phase function (and
incorporate the anisotropy factor g differently), relevant for different
tissues
    HENYEYGREENSTEIN=single(1);
    GEGENBAUERKERNEL=single(2);

    % note also of interest from a practical perspective is the "reduced
    % scattering coefficient" sometimes called effective isotropic
    % scattering coefficient, which is  $\mu_s*(1-g)$  and is the scattering
    % length it would have if g were zero... once the light is roughly
    % isotropic using the approximation of  $\mu_s*$  and ignoring phase
function
    % is fine, but when it's just coming out of a fiber or is otherwise
anisotropic, effective isotropic gives results showing the light "balling
    % up" too close to the fiber tip

```

```

%% create a lookup table for the CDF of the scattering angle as a
function of mu_s and g (for each tissue), as it's a slow calculation relative
to others in the MC code.
%%this process generates values theta as a function of a (0,1) uniform
random variable, for each value of g this is as suggested by delpy et al as a
BETTER WAY than solving for acos(cos(theta) where cos(theta) distribution is
not the right choice.
quadPoints=1001;
nRandVals=1e5;
nTissues=int32(size(tissueParam,1));
randThetas=zeros(nRandVals,nTissues);
eta=zeros(quadPoints,nTissues); %this will be the name of my uniform
random variable on (0,1)

for qqq=1:nTissues
    switch tissueParam(qqq,PHASE_FUNCTION)
        case HENYEYGREENSTEIN
            gg=tissueParam(qqq,ANISO_COEF);
            fun=@(ttheta) sin(ttheta)/2*(1-gg^2)./(1+gg^2-
2*gg*cos(ttheta)).^1.5; %this is the HG phase function, multiplied by 2pi
sin(theta) because the HG phase function is normalized over 4pi steradians,
so do the phi integral and multiply by sintheta to be ready for theta
integral just below
            eta(1,qqq)=0;
            for rrr=2:quadPoints %we are calculating the CDF so
the first point is (0,0)...start calculating from 2nd point
                eta(rrr,qqq)=integral(fun,0,(rrr-
1)/(quadPoints-1)*pi);
            end
            startsmooth= find(eta(:,qqq)>.9, 1 ); %smooth the
data above eta==.9 this is pretty flat (and very smooth), but noisy when
numerically integrated

            eta(startsmooth:quadPoints,qqq)=smooth(eta(startsmooth:quadPoints,qqq),
.05,'loess'); %%because the tail is so flat, numerical noise can cause the
eta plot to repeat values as it flattens, and repeated values cause interp1
to vomit on the next line. small values of eta rise quickly, meaning noise
doesn't create duplicate values, and smoothing only harms accuracy
            rrr=find(eta(:,qqq)>.998,1); %this part gets
extremely flat so numerical noise can lead to duplicate values which , AND
eta(quadPoints,qqq) <1 sometimes, and it NEEDS to be one or we'll get
problems when we get a random number very close to 1
            offby=1-eta(quadPoints,qqq);
            for sss=rrr:quadPoints
                eta(sss,qqq)=eta(sss,qqq)+offby*(sss-
rrr)/(quadPoints-rrr); %linearly interpolate the last few points so that eta
reaches 1 and makes a valid CDF, in spite of numerical integration noise
            end
        end
    end
end

```

```

        randThetas(:,qqq)=interp1(eta(:,qqq),0:pi/(quadPoints-
1):pi,0:1/(nRandVals-1):1,'spline','extrap'); %write in actual CDF of
scattered angles (theta)...get a theta by generating a rand on (0,1],
multiply by nRandVals and round it to an integer
        case GEGENBAUERKERNEL
            %this gets the gGK to use for making eta, and then
randThetas.
            gofgGK=zeros(2*quadPoints+1,1); %we want gGK as a function
of g, but this is what we can calculate directly. both function run -1 to 1
            gg=tissueParam(qqq,ANISO_COEF); %this is the value of g that
we actually want...once we have gGK(g), this is the value we'll use for g,
and apply that to get the theta distribution
            alphaGK=1; %this is fixed in the literature I am
using for these parameters, but in principle it can be varied.
            gofgGK(1)=-1;
            gofgGK(2*quadPoints+1)=1;
            for rrr=-quadPoints+1:quadPoints-1 %we skip the
endpoints because g(gGK==+/-1)==+/-1 and it doesn't calculate well so we
added it by hand
                if rrr==0
                    continue %the value gGK==g==0 is a
special case where p(theta,gGK) is NaN and actually requires a special form
of the phase function. skip this iteration as it's got the correct value (of
g=0) anyway.
                end
                gGK=rrr/quadPoints;
                %the next line is
                %2pi*sin(theta)*cos(theta)*p(theta,gGK), where
                %sin(theta) is for integrating sin(theta)
dtheta, and
                %cos(theta)*p(theta) is because we want the
                %expectation value of cos(theta)...that's the
                %definition of g. the phi integral yields a
2pi
                intToGetg=@(ttheta)
2.*pi.*sin(ttheta).*cos(ttheta).*(alphaGK.*gGK./(pi.*((1+gGK).^(2.*alphaGK)-
(1-gGK).^(2.*alphaGK))).*(1-gGK.^2).^(2.*alphaGK)./(1+gGK.^2-
2.*gGK.*cos(ttheta)).^(alphaGK+1));
            gofgGK(rrr+quadPoints+1)=integral(intToGetg,0,pi); %g is the mean value of
cos theta, so integrate over all theta to get that
            end
            gGKofg=interp1(gofgGK,-1:1/quadPoints:1,-
1:1/quadPoints:1,'spline');

            theRightgGK=gGKofg(round(gg*quadPoints)+quadPoints+1);

            %from here out, it's like the HENYEYGREENSTEIN case just with
a

```

```

%different function

    fun=@(ttheta)
2.*pi.*sin(ttheta).*(alphaGK.*theRightgGK./(pi*((1+theRightgGK).^2.*alphaGK)
-(1-theRightgGK).^2.*alphaGK))).*(1-
theRightgGK.^2).^2.*alphaGK./(1+theRightgGK.^2-
2.*theRightgGK.*cos(ttheta)).^(alphaGK+1));
        eta(1,qqq)=0;
        for rrr=2:quadPoints %we are calculating the CDF so
the first point is (0,0)...start calculating from 2nd point
            eta(rrr,qqq)=integral(fun,0,(rrr-
1)/(quadPoints-1)*pi);
        end
        startsmooth= find(eta(:,qqq)>.9, 1 ); %smooth the
data above eta==.9 this is pretty flat (and very smooth), but noisy when
numerically integrated

        eta(startsmooth:quadPoints,qqq)=smooth(eta(startsmooth:quadPoints,qqq),
.05,'loess'); %%because the tail is so flat, numerical noise can cause the
eta plot to repeat values as it flattens, and repeated values cause interp1
to vomit below. small values of eta rise quickly, meaning noise doesn't
create duplicate values, and smoothing messes only harms accuracy
        rrr=find(eta(:,qqq)>.998,1); %this part gets really
flat so numerical noise can lead to duplicate values, AND eta(quadPoints,qqq)
<1 sometimes, and it NEEDS to be one or we'll get problems when we get a
random number very close to 1
        offby=1-eta(quadPoints,qqq);
        for sss=rrr:quadPoints
            eta(sss,qqq)=eta(sss,qqq)+offby*(sss-
rrr)/(quadPoints-rrr); %linearly interpolate the last few points so that eta
reaches 1 and makes a valid CDF, in spite of numerical integration noise
        end

        randThetas(:,qqq)=interp1(eta(:,qqq),0:pi/(quadPoints-
1):pi,0:1/(nRandVals-1):1,'spline','extrap'); %write in a line of the
thetas...get a theta by generating a rand on (0,1], multiply by nRandVals and
round it to an integer

    end

end

    randThetas=single(randThetas); %above was calculated with double
precision to cut down on numerical errors, especially associated with very
small deflections when scattering with theta~0, which is quite frequent in
tissue. convert to single for GPU use

%% brain geometry

```

```

    blocksize=s.blocksize; %size of tissue to be modeled (3 element vector,
in mm)
    vox=s.vox; %size of each voxel (3 element vector, in mm)
    nvox=s.nvox; %number of voxels in each dimension (3 element vector,
dimensionless)

    %location, direction, size of detection fiber tip,and its half
    %acceptance angle. Note this is not the fiber delivering excitation
    %light
    tipPos=s.receiveTipPos;
    tipDiam=s.receiveTipDiam;
    tipNorm=s.receiveTipNorm;
    tipAA=s.receiveTipAA;

    flux=zeros(nvox(X),nvox(Y),nvox(Z),'single'); %this will store the flux
through each voxel in the brain

    %% create the starting conditions for each photon we want to run (ie, the
light sources)
    %%photonPhase is an nphotons x 8 array, where the first 6 elements are
x,y,z and
    %%vx, vy, vz where the velocity is normalized velocity (ie, direction)
of
    %%propagation. ie, it is a point in the 6N dimensional phase space
    %% the 7th entry is the weight, which drops as the photon packet is
    %% absorbed while propagating. the 8th entry is to pad it to 32 bytes
for better CUDA performance,
    %% and is unused
    photonPhase=zeros(nphotons,8,'single');
    photonFlag=true(nphotons,1); %if this flag is set, the photon must be
calculated. if its weight is absorbed to nothing or it hits boundary, this
is set to zero and photon is not advanced
    %this can definitely be moved to GPU if it is a bottleneck (it is at
    %least a bit), but makes the code somewhat harder to read/use

%% code cell for doing illumination from fiber tip.

    rmax=s.sendTipDiam/2; %RADIUS of fiber in mm
    randr=sqrt(rand(nphotons,1,'single')*rmax^2);
    randp=2*pi*rand(nphotons,1,'single');
    photonPhase(:,Y)=s.sendTipPos(2)+randr.*sin(randp);
    photonPhase(:,Z)=s.sendTipPos(3)+randr.*cos(randp);
    clear randr randp
    photonPhase(:,X)=s.sendTipPos(1);

```

```

    halfOpenAngle=s.sendTipAA; %this is the HALF open angle in degrees of
a uniform emission of photons from a fiber tip (approximation to a laser
coupled to the fiber)
    theta=single(acos(1-rand(nphotons,1)*(1-cos(halfOpenAngle)))); %this
gets theta between 0 and 15 degrees. NOTE: I once had matlab's cosd function
PRODUCE WRONG VALUES. can't reproduce but it was DEFINITELY real. restart
of matlab fixed.
    %theta=single(acos(1-2*rand(nphotons,1))); %this would produce random
direction for photons
    phi=single(rand(nphotons,1)*2*pi);
    photonPhase(:,VX)=cos(theta); %x coordinate of direction of travel
    photonPhase(:,VY)=sin(theta).*sin(phi);
    photonPhase(:,VZ)=sin(theta).*cos(phi);
    photonPhase(:,WEIGHT)=1; %the weight of all photons start at 1.
    % if the fiber is tilted, we need to adjust the location and direction
    % of photons accordingly.
    thetaOrient=acos(s.sendTipNorm(1));
    if sin(thetaOrient)~=0
        phiOrient=acos(s.sendTipNorm(3)/sin(thetaOrient));
        if s.sendTipNorm(2)<0 %acos is defined over 0 to pi but phi must
cover 0 to 2pi, or more conveniently -pi to pi; use sign of y component to
see if negative sign needed
            phiOrient=-phiOrient;
        end
    else
        phiOrient=0; %it's degenerate and irrelevant when fiber is straight
up/down, and the other branch of the if will lead to divide by zero
    end
    %now do the angle transform of position and direction using sendtippos
%as origin for position. note I do not reconcile all my axes with
%conventional euler angles, but this is such a transform, using only
%two of the three angles as I am assuming a cylindrical fiber that
%would be invariant under the final rotation.
    %theta is a _clockwise_ rotation about the y axis, phi is a _clockwise_
%rotation about the original x axis.
    rotmat=[cos(thetaOrient),0,-
sin(thetaOrient);sin(phiOrient)*sin(thetaOrient),cos(phiOrient),sin(phiOrient
)*cos(thetaOrient);cos(phiOrient)*sin(thetaOrient),-
sin(phiOrient),cos(thetaOrient)*cos(phiOrient)];

    for ii=1:nphotons
        photonPhase(ii,1:3)=(photonPhase(ii,1:3)-
s.sendTipPos)*rotmat'+s.sendTipPos; %positions are row vectors in
photonPhase, so do transpose of rotmat*(columnvectorofposition)
        photonPhase(ii,4:6)=photonPhase(ii,4:6)*rotmat';
    end

    incVar=uint32(0);

```

GPUincVar=gpuArray(incVar); %this is incremented atomically so we know for sure when all the photons have been calculated and we can sum results and such

```
%load the appropriate CUDA kernel and pass the needed data to GPU
%memory. Note newer versions of matlab
%incorporate more user-friendly access to GPU acceleration directly in
matlab,
%and this method of writing and calling a CUDA kernel is no longer
%recommended in many circumstances
if s.sidesrefl==true
```

```
k=parallel.gpu.CUDAKernel('photonpropagatorwithfibersidesreflect.ptx','photon
propagatorwithfibersidesreflect.cu','photonpropagator');
else
```

```
k=parallel.gpu.CUDAKernel('photonpropagatorwithfiber.ptx','photonpropagatorwi
thfiber.cu','photonpropagator');
end
```

```
tbs=k.MaxThreadsPerBlock;
k.ThreadBlockSize=tbs;
k.GridSize=double(ceil(double(nphotons)/tbs));
GPUPhotonPhase=gpuArray(photonPhase);
GPUFluxLocation=gpuArray(zeros(nphotons,4,'single'));
GPUFlux=gpuArray(flux);%in the end this is where the fluence
(isotropic flux) goes...GPUFluxLocation is a way to asynchronously generate
each flux before one thread adds it into the matrix
GPURandThetas=gpuArray(randThetas);
GPUPhotonFlag=gpuArray(photonFlag);
test=zeros(nphotons,1,'single');
currVox=zeros(nphotons,3,'int32');
for qqq=1:nphotons
    currVox(qqq,:)=floor(photonPhase(qqq,1:3)./vox); %use FLOOR here
to get zero based currVox for the cuda code, which is zero based ala c
end
GPUCurrVox=gpuArray(currVox);
stillRunningFlag=true;
qq=0; %incrementing variable
tic
flagsFlipped=false(nphotons,1);
nleft=zeros(10000,1);
```

```
while stillRunningFlag
    %call the CUDA code to advance every photon up through a
    %scattering event or entering a new voxel, recording how much
    %light passed through a voxel and partially absorbing each
    %photon. note this means each step is not a fixed advance in
    %time, but rather a step to the next event that affects the
```

```

%photon.

rands=rand(nphotons,4,'single');
GPURands=gpuArray(rands);

[test,GPUPhotonPhase,GPUFlux,GPUCurrVox,
GPUPhotonFlag]=feval(k,test,GPUPhotonPhase, GPUFlux, GPUCurrVox,
GPUPhotonFlag, GPUincVar, GPUFluxLocation, scatter, absorb, g, GPURands, vox,
nvox, nphotons, GPURandThetas, int32(nRandVals),
tipPos,tipDiam,tipNorm,tipAA);

qq=qq+1;

photonFlag=gather(GPUPhotonFlag);
qqq=sum(photonFlag)
nleft(qq)=qqq;
if(qqq<nphotons/1e4) % a few photons will sometimes get "stuck"
and not compute further due to some edge cases I haven't fully pinned down.
when 99.99% of photons are done (ie, fully absorbed or exited simulated
volume) call it a day and stop
    stillRunningFlag=false;
end
    photonPhase=gather(GPUPhotonPhase);
    photonFlag(isnan(photonPhase(:,X)))=false; %Due to numerical
errors in the single precision calculations on the GPU, once in a great while
the CUDA code will attempt to do acos(1.00000001) or similar and return NAN
(possibly for several components of photon's phase space, but if it is nan
for any, it is nan for X so I only check that)
    flagsFlipped(isnan(photonPhase(:,X)))=true;
end
toc
outFlux=double(gather(GPUFlux));
outPhotonPhase=photonPhase;
outNVox=nvox;
outVox=vox;
outBlockSize=blocksize;
delete(k);
%%optional potential speedup (actually I think it'll slow) is to pass
the
%%photons to the cuda kernel with the mask of photonFlag==true so you
%%don't try to calculate photons that are done...still, gpu is so fast
%%that little short circuit calculation is probably faster than doing
extra matlab ops.

end

```

photonpropagatorwithfiber.cu

```
#include <stdio.h>
#include <stdlib.h>
#include <cuda.h>
#include <curand_kernel.h>
#include <math.h>

//make macros for array index referencing
#define GRIDOFFSETD(whichrow,whichcol,nrows) ((whichrow) +
(whichcol)*(nrows))
#define VOXELOFFSETD(whichx,whichy,whichz,nx,ny)
((whichx)+(whichy)*(nx)+(whichz)*(nx)*(ny))
#define X 0
#define Y 1
#define Z 2
#define VX 3
#define VY 4
#define VZ 5
#define WEIGHT 6 //for photonPhase, the 7th element is that photon's weight
#define FLUX 3 //for fluxlocation, the 4th element is the quantity of flux in
that location

__global__ void photonpropagator( float * test, float * photonPhase, float *
flux, int * currVox, bool * photonFlag, unsigned int * incVar, float *
fluxLocation, const float * scatter, const float * absorb, const int * g,
const float * rands, const float * vox,const int * nvox,const int nphotons,
const float * randThetas, const int nRandVals, const float * tipPos, const
float tipDiam, const float * tipNorm, const float tipAA)
{
    float pi=3.14159265358979;
    float killThreshold=0.0001; //once weight reaches .0001, regenerate
photon to weight=1 with probability .0001 otherwise stop calculating the
photon. conserves weight correctly
    int idx=threadIdx.x;
    int i=blockIdx.x*blockDim.x+idx; //this should be the index of the
photon, running from 1 to nphotons. each i will be run once, but not
necessarily in order
    float voxelVol=vox[X]*vox[Y]*vox[Z];
    if (i >= nphotons) {return;} //the divergence should occur only in the
final warp; not a big performance hit
    fluxLocation[GRIDOFFSETD(i,FLUX,nphotons)]=0.f;// this zeros the flux
of each photon from the last timestep. it's updated down below, but NOT if
the photon has reached the boundary
    if (photonFlag[i]) { //if this photon is still flagged as needing to
be propagated, do this. otherwise it'll skip to the atomicInc part at the end
        //figure out which face of the voxel you will hit, without using
branching control structures that split warps and ruin parallelization
```

```

//dx is the TOTAL distance traveled to reach the boundary of the
voxel in the x plane, and so forth.
float dx=((currVox[GRIDOFFSETD(i,X,nphotons)] + 1.f -
signbit(photonPhase[GRIDOFFSETD(i,VX,nphotons)]))*vox[X]-
photonPhase[GRIDOFFSETD(i,X,nphotons)]/photonPhase[GRIDOFFSETD(i,VX,nphotons
)];
float dy=((currVox[GRIDOFFSETD(i,Y,nphotons)] + 1.f -
signbit(photonPhase[GRIDOFFSETD(i,VY,nphotons)]))*vox[Y]-
photonPhase[GRIDOFFSETD(i,Y,nphotons)]/photonPhase[GRIDOFFSETD(i,VY,nphotons
)];
float dz=((currVox[GRIDOFFSETD(i,Z,nphotons)] + 1.f -
signbit(photonPhase[GRIDOFFSETD(i,VZ,nphotons)]))*vox[Z]-
photonPhase[GRIDOFFSETD(i,Z,nphotons)]/photonPhase[GRIDOFFSETD(i,VZ,nphotons
)];

//this selects the shortest distance without using any control
branching. this is the distance it will take to reach the voxel boundary in
the current direction of travel.
float
dshortest=dx*(dx<dy)*(dx<dz)+dy*(dy<dx)*(dy<dz)+dz*(dz<dx)*(dz<dy);

//calculates a distance to travel before scattering, from the
appropriate exponential distribution
float dscat=-
logf(rands[GRIDOFFSETD(i,0,nphotons)]/scatter[VOXELOFFSETD(currVox[GRIDOFFSE
TD(i,X,nphotons)],currVox[GRIDOFFSETD(i,Y,nphotons)],currVox[GRIDOFFSETD(i,Z,
nphotons)],nvox[X],nvox[Y]]);

//if the distance before scattering is greater than distance to
the edge of the voxel, move to the edge of the voxel and stop (after all, the
next voxel might have different scattering properties, and if not you will
still get the right distribution because of the "memoryless" property of
exponential dsitribs
//else move to the point where you scatter and calculate your new
direction.

float dtip=fabsf((photonPhase[GRIDOFFSETD(i,X,nphotons)]-
tipPos[X])*tipNorm[X]+(photonPhase[GRIDOFFSETD(i,Y,nphotons)]-
tipPos[Y])*tipNorm[Y]+(photonPhase[GRIDOFFSETD(i,Z,nphotons)]-
tipPos[Z])*tipNorm[Z]); //this first line is distance to nearest point on the
plane
dtip /= -
(photonPhase[GRIDOFFSETD(i,VX,nphotons)]*tipNorm[X]+photonPhase[GRIDOFFSETD(i
,VY,nphotons)]*tipNorm[Y]+photonPhase[GRIDOFFSETD(i,VZ,nphotons)]*tipNorm[Z])
; //this modifies dtip to be distance to point on plane photon actually hits,
as it won't be flying directly at plane. if dtip is negative, it is heading
away from the plane

```

```

        float
TX=photonPhase[GRIDOFFSETD(i,X,nphotons)]+dtip*photonPhase[GRIDOFFSETD(i,VX,n
photons)]; //this is a temp value for where photon would hit plane of fiber
tip
        float
TY=photonPhase[GRIDOFFSETD(i,Y,nphotons)]+dtip*photonPhase[GRIDOFFSETD(i,VY,n
photons)];
        float
TZ=photonPhase[GRIDOFFSETD(i,Z,nphotons)]+dtip*photonPhase[GRIDOFFSETD(i,VZ,n
photons)];

        float asq=powf((TX-tipPos[X]),2)+powf((TY-tipPos[Y]),2)+powf((TZ-
tipPos[Z]),2); //when the photon is in the plane of the tip, how far from tip
center is this? compare to tip radius to decide if it has hit
        float
vdn=photonPhase[GRIDOFFSETD(i,VX,nphotons)]*tipNorm[X]+photonPhase[GRIDOFFSET
D(i,VY,nphotons)]*tipNorm[Y]+photonPhase[GRIDOFFSETD(i,VZ,nphotons)]*tipNorm[
Z]; //get velocity dot normal which we can compare to cos(tip acceptance
angle) to decide if the photon would be collected by the fiber
        if ((dtip>0) && (dtip<dscat) && (dtip<dshortest) &&
(asq<powf(tipDiam,2)/4) && (vdn<-cosf(tipAA))) {
            float
absorbVal=absorb[VOXELOFFSETD(currVox[GRIDOFFSETD(i,X,nphotons)],currVox[GRID
OFFSETD(i,Y,nphotons)],currVox[GRIDOFFSETD(i,Z,nphotons)],nvox[X],nvox[Y])];
//this is just shorthand for the absorption coefficient in the voxel, which
is tedious to reference

                photonPhase[GRIDOFFSETD(i,WEIGHT,nphotons)] *= expf(-
absorbVal*dshortest);

            photonPhase[GRIDOFFSETD(i,X,nphotons)]+=photonPhase[GRIDOFFSETD(i,VX,np
hotons)]*dtip;

            photonPhase[GRIDOFFSETD(i,Y,nphotons)]+=photonPhase[GRIDOFFSETD(i,VY,np
hotons)]*dtip;

            photonPhase[GRIDOFFSETD(i,Z,nphotons)]+=photonPhase[GRIDOFFSETD(i,VZ,np
hotons)]*dtip;
            photonFlag[i]=0;
        } else {

            if (dscat>dshortest) { //this is a flat out branch that will
probably split a warp. can avoid only(?) by calling lots of equations with
half of them multiplied by (dshortest<=dvox) and the other half by
(dshortest>dvox)

                float
absorbVal=absorb[VOXELOFFSETD(currVox[GRIDOFFSETD(i,X,nphotons)],currVox[GRID
OFFSETD(i,Y,nphotons)],currVox[GRIDOFFSETD(i,Z,nphotons)],nvox[X],nvox[Y])];

```

```

//this is just shorthand for the absorption coefficient in the voxel, which
is tedious to reference
    float
avgWeight=photonPhase[GRIDOFFSETD(i,WEIGHT,nphotons)]*(1.f-expf(-
absorbVal*dshortest))/absorbVal/voxelVol; //this is the distance-weighted,
volume normalized average flux. This is the contribution to the fluence in
this voxel from this photon.
    //add the flux for this photon, which gets inserted into the
flux 3d array at the end to prevent threads from overwriting each other in
that array

fluxLocation[GRIDOFFSETD(i,X,nphotons)]=currVox[GRIDOFFSETD(i,X,nphotons)];

fluxLocation[GRIDOFFSETD(i,Y,nphotons)]=currVox[GRIDOFFSETD(i,Y,nphotons)];

fluxLocation[GRIDOFFSETD(i,Z,nphotons)]=currVox[GRIDOFFSETD(i,Z,nphotons)];

fluxLocation[GRIDOFFSETD(i,FLUX,nphotons)]=avgWeight;//(photonPhase[GRIDOFFSE
TD(i,WEIGHT,nphotons)]-remainingWeight)/dshortest/absorbVal; //this is the
average value of the photon's weight in the voxel, which indicates its
contribution to the average flux in the voxel...a bit of a slippery quantity,
but not misleading

    //then advance its position and update its weight. note
velocities don't change because there was no scattering
    photonPhase[GRIDOFFSETD(i,WEIGHT,nphotons)] *= expf(-
absorbVal*dshortest);

photonPhase[GRIDOFFSETD(i,X,nphotons)]+=photonPhase[GRIDOFFSETD(i,VX,nphotons
)]*dshortest;

photonPhase[GRIDOFFSETD(i,Y,nphotons)]+=photonPhase[GRIDOFFSETD(i,VY,nphotons
)]*dshortest;

photonPhase[GRIDOFFSETD(i,Z,nphotons)]+=photonPhase[GRIDOFFSETD(i,VZ,nphotons
)]*dshortest;

    //update to the new value of currvox. currvox is useful to
have stored because you don't have to calculate which voxel you're in via its
location, which is currently right on a border and subject to float math
gotchas
    currVox[GRIDOFFSETD(i,X,nphotons)]+=(1-
2*signbit(photonPhase[GRIDOFFSETD(i,VX,nphotons)]))*(dx<dy)*(dx<dz);
    currVox[GRIDOFFSETD(i,Y,nphotons)]+=(1-
2*signbit(photonPhase[GRIDOFFSETD(i,VY,nphotons)]))*(dy<dx)*(dy<dz);
    currVox[GRIDOFFSETD(i,Z,nphotons)]+=(1-
2*signbit(photonPhase[GRIDOFFSETD(i,VZ,nphotons)]))*(dz<dy)*(dz<dx);

```

```

        //handle boundary conditions
        //this is for absorbing boundary conditions
        //set the flag to 0 if the photon has entered a no-go voxel,
so it will not be calculated in any further steps.
        photonFlag[i] = !(currVox[GRIDOFFSETD(i,X,nphotons)] < 0 ||
currVox[GRIDOFFSETD(i,X,nphotons)] >= nvox[X] ||
currVox[GRIDOFFSETD(i,Y,nphotons)]<0 || currVox[GRIDOFFSETD(i,Y,nphotons)] >=
nvox[Y] || currVox[GRIDOFFSETD(i,Z,nphotons)]<0 ||
currVox[GRIDOFFSETD(i,Z,nphotons)] >= nvox[Z]);

    } else { //if dscat>dshortest
        int scatCurrx=currVox[GRIDOFFSETD(i,X,nphotons)]; //local
variables to shorten calls to VOXELOFFSETD
        int scatCurry=currVox[GRIDOFFSETD(i,Y,nphotons)];
        int scatCurrz=currVox[GRIDOFFSETD(i,Z,nphotons)];
        int
localg=g[VOXELOFFSETD(scatCurrx,scatCurry,scatCurrz,nvox[X],nvox[Y])]-1;
//the index of the randthetas lookup table for the relevant voxel... it is
NOT the value of g. The -1 is to shift to zero-based c from 1 based matlab
        float
absorbVal=absorb[VOXELOFFSETD(currVox[GRIDOFFSETD(i,X,nphotons)],currVox[GRID
OFFSETD(i,Y,nphotons)],currVox[GRIDOFFSETD(i,Z,nphotons)],nvox[X],nvox[Y])];
//this is just shorthand for the absorption coefficient in the voxel, which
is tedious to reference
        float
avgWeight=photonPhase[GRIDOFFSETD(i,WEIGHT,nphotons)]*(1.f-expf(-
absorbVal*dscat))/absorbVal/voxelVol; //this is the distance-weighted,
volume normalized average flux. ask mike for more details.

fluxLocation[GRIDOFFSETD(i,X,nphotons)]=currVox[GRIDOFFSETD(i,X,nphotons)];
fluxLocation[GRIDOFFSETD(i,Y,nphotons)]=currVox[GRIDOFFSETD(i,Y,nphotons)];
fluxLocation[GRIDOFFSETD(i,Z,nphotons)]=currVox[GRIDOFFSETD(i,Z,nphotons)];
        fluxLocation[GRIDOFFSETD(i,FLUX,nphotons)]=avgWeight;

photonPhase[GRIDOFFSETD(i,X,nphotons)]+=photonPhase[GRIDOFFSETD(i,VX,nphotons
)]*dscat; //move the photon the distance it traveled before scattering

photonPhase[GRIDOFFSETD(i,Y,nphotons)]+=photonPhase[GRIDOFFSETD(i,VY,nphotons
)]*dscat;

photonPhase[GRIDOFFSETD(i,Z,nphotons)]+=photonPhase[GRIDOFFSETD(i,VZ,nphotons
)]*dscat;
        photonPhase[GRIDOFFSETD(i,WEIGHT,nphotons)] *= expf(-
absorbVal*dscat);

```

```

        int
pickThisTheta=floorf(rands[GRIDOFFSETD(i,1,nphotons)]*nRandVals); //this is
the photon's very own random number
        float
xi=randThetas[GRIDOFFSETD(pickThisTheta,localg,nRandVals)];//this is some
random xi from the distribution...many photons can get the same one.
        float psi=2.f*pi*rands[GRIDOFFSETD(i,2,nphotons)];

        float tempVX=photonPhase[GRIDOFFSETD(i,VX,nphotons)]; //this
is a shorter reference to the current values, for use in calculating the new
values below
        float tempVY=photonPhase[GRIDOFFSETD(i,VY,nphotons)];
        float tempVZ=photonPhase[GRIDOFFSETD(i,VZ,nphotons)];
//we will call the unscattered vector (0,0,1) in some unknown
frame. we will rotate it around the x axis by beta radians
//then the z axis by alpha radians (both axes in that unknown
but fixed reference frame). this will get us our vector in our known
cartesian coord system.

//we now do the scattering so our photon is now in the
scattered direction (sin(xi)*sin(psi),sin(xi)*cos(psi),cos xi)
//in our abstracted coordinates. we then do the rotations
about alpha and beta described above, and we end
//up with scattered photon in original cartesian coordinates
(nb this only takes two angles because rotating about
//the photon's velocity vector doesn't do anything)
//this code gets tedious to do step by step so I'm just going
to write it all down. basic matrix rotations gets you here.
//we get GIMBAL LOCK if the vector in the original frame is
right on z direction (and the generic equation divides by zero). whichever
direction we do that first rotation around will be trouble always.

        if (sqrtf((1.f-tempVZ)*(1.f+tempVZ))>0) { //this branch will
be used almost all the time (eg 99999/100000) so not much warp splitting
//scatter the velocity from the old values (stored in
temp) to the new values
//sin(alpha)=VX/sqrt(1-VZ^2), cos(alpha)=-VY/sqrt(1-
VZ^2),cos(beta)=VZ,sin(beta)=sqrt(1-VZ^2) the squares are done (1-VZ)*(1+VZ)
in an effort to minimize float issues.
//note the ULPs for each operation are listed in the cuda
programming guide.
        float sinalpha=tempVX/sqrtf((1.f-tempVZ)*(1.f+tempVZ));
        float sinbeta=sqrtf((1.f-tempVZ)*(1.f+tempVZ));
        float cosalpha=-tempVY/sqrtf((1.f-tempVZ)*(1.f+tempVZ));
        float cosbeta=tempVZ;

photonPhase[GRIDOFFSETD(i,VX,nphotons)]=cosf(psi)*sinf(xi)*cosalpha -
sinalpha*cosbeta*sinf(xi)*sinf(psi) + sinalpha*sinbeta*cosf(xi);

```

```

photonPhase[GRIDOFFSETD(i,VY,nphotons)]=sinalpha*sinf(xi)*cosf(psi) +
cosalpha*cosbeta*sinf(xi)*sinf(psi) - cosalpha*sinbeta*cosf(xi);

photonPhase[GRIDOFFSETD(i,VZ,nphotons)]=sinbeta*sinf(xi)*sinf(psi) +
cosbeta*cosf(xi);

        } else { //otherwise the photon is gimbal locked along z,
but this is easy to scatter "by hand"

photonPhase[GRIDOFFSETD(i,VX,nphotons)]=sinf(xi)*cosf(psi);

photonPhase[GRIDOFFSETD(i,VY,nphotons)]=sinf(xi)*sinf(psi);
        photonPhase[GRIDOFFSETD(i,VZ,nphotons)]=cosf(xi)*tempVZ;
//mult by tempVZ (which is +/- 1) gets us correct sign for Z...is the photon
scattering from straight up or straight down?

        }
    } //else clause of if dscat>dshortest
} //else clause of if (dtip<dscat....)
//run a check on photons whose weight is dwindled to a fraction
of a percent: eliminate them for efficiency but to conserve weight, add in
the change that they'll be regenerated to weight=1. do it without branching!
float weightAtKill=photonPhase[GRIDOFFSETD(i,WEIGHT,nphotons)];
//the next line leaves weight as it is if the weight is over
killThreshold, or sets it equal to -(-1*(0 with prob 1-killthreshold or 1
with prob killThreshold)
//note this formulation will be bad times if weight is exactly 1,
which it will NOT be by the end of this code; either scattered or not, it
will have had some absorbtion.
photonPhase[GRIDOFFSETD(i,WEIGHT,nphotons)]=ceilf(weightAtKill-
killThreshold)*weightAtKill - floorf(weightAtKill-
killThreshold)*(rands[GRIDOFFSETD(i,3,nphotons)] < weightAtKill);
//if the weight was set to exactly zero in the last step, we
should kill the photon obviously.

photonFlag[i]=photonFlag[i]*(photonPhase[GRIDOFFSETD(i,WEIGHT,nphotons)
]!=0);

    }//if (photonFlag[i]

//always increment the incVar, which *would be* equal to i if the
blocks executed sequentially which they may not. maxes out at nphotons-1 due
to being zero based
//last thread left apply the flux and turn out the lights
if(atomicInc(incVar,nphotons-1)==nphotons-1) { //this only occurs
once, so not a big warp performance hit, but is rather non-parallelizable.
TODO should maybe do atomicAdd's instead??
int currx,curry,currz;

```

```

        for (int j=0;j<nphotons;j++) {
            currx=fluxLocation[GRIDOFFSETD(j,X,nphotons)];
            curry=fluxLocation[GRIDOFFSETD(j,Y,nphotons)];
            currz=fluxLocation[GRIDOFFSETD(j,Z,nphotons)];

            flux[VOXELOFFSETD(currx,curry,currz,nvox[X],nvox[Y])]+=fluxLocation[GRI
DOFFSETD(j,FLUX,nphotons)];
        }
        *incVar=0; //reset the incVar to zero for the next step of the
photon propagation
    }
}

```

B. Implantable light field microimager simulation code

mpr_param.m

```

% Helper function for whole_enchilada.
% Last update 5-26-2017 to add this comment
% Mike Henninger.
function mpr=mpr_param(cs_basename)
mpr.version='3.1.0';
%mpr_param_timeseries
%provides parameters to timeseriesgenerator2 (or whole_enchilada or
elsewhere)
%to make it much more easily rolled into a script
%if you want to roll everything up into a single parameter struct to make
%multipdwrap a function, search and replace ;;; with mpr_2012. and that
%will fix all of the left hand assignments, leaving just a couple of broken
%right hand equations that will complain cuz their vars won't be defined.
%also see end of this file
global cspath
global sensor_orientations

mpr.CS_basename=cs_basename; %cell array of strings for each basename

%params for generating video and its reconstruction:
mpr.framerate=150; %frames per second recorded by sensor
mpr.duration=9; %duration of video recorded by sensor, in seconds
mpr.spikerate=4; %spike rate of neurons in hz
mpr.deltaF_overF=.19; %roughly what GCaMP6f looks like, per Fast calcium
sensor proteins for monitoring neural activity Badura et al.
mpr.baseline_fluorescence=1;

%params for simulating neurons
mpr.neuronbox=[-.045,.045,-.095,.095,.005,.095]; %this is the volume for
which we are simulating neurons, w.r.t. the center of the photon

```

```

simulated volume (the 4x4x2 box)
%mpr.neurondensity=1/((mpr.neuronbox(2)-mpr.neuronbox(1))*(mpr.neuronbox(4)-
mpr.neuronbox(3))*(mpr.neuronbox(6)-mpr.neuronbox(5))); %neurons per
cubic mm, mouse cortex is about 10^5. this is set to 1, like timeseries
likes it.
mpr.neuron_prefix='uniform'; %this is the prefix at the start of the file
name for loading simulated neurons... values are "shell" and "uniform"
as of 5-29-2015
mpr.neuron_basename='y2000z2000at532nm.mat'; %this is the end of the file
name used for loading simulated neurons.
mpr.placemode=2; %1=random locations, using density and location parameters
mentioned above. 2=hand picked locations
mpr.desired_neuron_density=1e5; %neurons/mm^3. full cortical density is 1e5
mpr.position_error_type='directional'; %are the position errors of neurons
randomly distributed or directional? if directional, use the below line
for direction (ignored for random)
mpr.position_error_magnitude=0; % in mm
mpr.position_error_direction=[0,0,1]; %normalized direction vector

mpr.desiredx=[];%input as row vectors for convenience, but convert to column
vectors in multipdwrap. one entry per neuron you wish to sim
mpr.desiredy=[]; %desiredx desiredy desiredz need to be the same length
mpr.desiredz=[];
%mpr.spikeshape=single([zeros(1,2920),0:1/80:1,exp(-(1:3000)/610)]); %this is
plotted in ms. roughly what gcamp3 looks like
mpr.risetime=45; %in ms, time from baseline to peak
mpr.thalf=142; %in ms, time from peak to half of peak
mpr.spikeshape=single([zeros(1,2930),0:1/mpr.risetime:1,exp(-
(1:3000)/(mpr.thalf/log(2)))]); %this is plotted in ms. roughly what
GCaMP6f looks like, per Fast calcium sensor proteins for monitoring
neural activity Badura et al.
mpr.originalneuronlocation=[2,2];%this is the xy location of where the neuron
was simulated...the canonical is at 2,2,neurdist in a 4x4x2 mm box
mpr.neuronspacing=.002; %this is spacing in the z (perp to imager) dimension,
which is just the spacing at which I simulated the neurons. XY can just
be linearly translated; Z needs to be re-simulated
%mpr.nbgphotons=1e6; %number of photons used to simulate each 200um x 200um x
10um slab of diffuse fluorescence from the neuropil
mpr.nfgphotons=4e7; %number of photons used to simulate each spherical
"neuron"
mpr.neuron_diameter=5; %neuron diameter in um...this is not a free parameter;
it should agree with what is simulated

%params for lighting, shot noise, and background
mpr.lightingvoxel=[10 10 10];
mpr.lightingvol=[-295 295 -295 295 5 495]; %as with neuronbox and csvol,
0,0,0 is the center of the z=0 face of the volume. if the lighted volume
is offset, just add constants here. lightedvol is defined so that the z
direction is in +zhat of the "lab coordinates"

```

```

mpr.lightingarray=ones(round((mpr.lightingvol(2)-
    mpr.lightingvol(1))/mpr.lightingvoxel(1)),round((mpr.lightingvol(4)-
    mpr.lightingvol(3))/mpr.lightingvoxel(2)),round((mpr.lightingvol(6)-
    mpr.lightingvol(5))/mpr.lightingvoxel(3)),'single');
mpr.lightingcutoff=1e-3; %if the value of lightingarray is less than this
    value at a neuron, that neuron is consider not illuminated and not
    included in the post-hoc neuron locations. Note that such neurons are
    still in the outshiftvec vectors.
mpr.fgbrightness=1; %the ratio of a pixel of cell body to a pixel of neuropil
    is fgbrightness+1 (the background is everywhere including cell body!)
mpr.shotnoise=true;
mpr.background=false; %should we include background light (due to
    fluorophores in unresolvable cell processes, autofluorescence, etc.)
mpr.sensornoise=true; %added in v.2.6, has conversion from photons to
    electrons, sensor noise, and digitization noise.
mpr.nsenors=numel(mpr.CS_basename); %the number of sensors being simulated
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sensorindex=1; % this is a counter increment for each sensor. it's like a big
    unrolled loop, with a bunch of assignments for each sensor
%to use, copy and paste all the sensor parameters, make the changes for the
    additional sensor, and increase sensorindex between the blocks
%
    mpr.CS_basename{sensorindex}='_1p3um_80x200_by_4um_400x200x200_sd2p7_OF5
    0pct_532_slope015_1'; %this string, prefixed with 'A' or 'mask' loads
    the CS variables from disk

mpr.sensorpos(sensorindex,:)=1,1]; %where is the top left corner of the
    sensor, in terms of the coordinates used by photonPhaseTotal
mpr.dsensors(sensorindex)=sensor_orientations.zhat; %grab the sensor
    orientation from the global enum struct, one for each sensor (natch)
mpr.psensors(sensorindex,:)=0,0,0]; %vector in units of mm indicating the
    position of the center of the sensor with respect to the x=0,y=0,z=0
    point in the lighting volume, which corresponds to the center of the
    lighting volume
mpr.electronsPerPhoton(sensorindex)=.5224; %conversion from photon count
    hitting pixel to electrons generated in that pixel, based on sensitivity
    of 4800 e-/lx/s and QE of 0.6, from sukegawa 2013
mpr.readNoise(sensorindex)=2; %measured in electrons!
mpr.bitsPerPixel(sensorindex)=10; %assume the adc has 2^bitsPerPixel levels
    to assign to
mpr.wellDepth(sensorindex)=5e3; %in electrons...what's the max number of
    electrons we can count before saturation?
clear('ncsphotons','slope','NInner','NOuter') %gonna overwrite it below
    anyway, and if the var isn't there to overwrite we need to know that

load(strcat(cspath,'mask',mpr.CS_basename{sensorindex}))
%note this loads up mask AND px py 200. wy maskelementsiz
    sensordistance,nx,ny,nz overwriting some of the above initializations!
if exist('NInner','var')==1

```

```

    mpr.NInner{sensorindex}=NInner; %index of refraction below this mask
    (but above any additional masks if using multiple masks over one
    sensor...)
    mpr.NOuter{sensorindex}=NOuter;
else
    mpr.NInner{sensorindex}=1.37;
    mpr.NOuter{sensorindex}=1.37;
    warning('no indices of refraction found; assuming NOuter=NInner=1.37')
end
mpr.csvol(sensorindex,:)=csvol; clear csvol;
mpr.csvoxel(sensorindex,:)=csvoxel; clear csvoxel;
mpr.nx(sensorindex)=nx;clear nx;
mpr.ny(sensorindex)=ny;clear ny;
mpr.nz(sensorindex)=nz;clear nz;
mpr.px(sensorindex)=px;
mpr.py(sensorindex)=px;clear px; %DEBUG find the matlab namespace colision
    with py that makes this not work for =py. for all cases in the present
    paper px=py, so no problem, but the general case is bugged.
mpr.sensorpos(sensorindex,:)=sensorpos;clear sensorpos;
mpr.wx(sensorindex)=wx;clear wx;
mpr.wy(sensorindex)=wy;clear wy;
if exist('ncsphoton','var')
    mpr.ncsphoton(sensorindex)=ncsphoton;clear ncsphoton;
else
    fprintf('caution: ncsphoton was not specified in the mask file;
    assuming 4e7 (standard for 4um voxels)')
    mpr.ncsphoton(sensorindex)=4e7; %number of photons used to simulate
    each voxel for compressed sensing image spaces. these nphoton vars
    should match the simulations used; not a free parameter
end

if exist('sensor_category','var')
    mpr.sensor_category=sensor_category;clear sensor_category;
    switch mpr.sensor_category
        case 'masked'
            if iscell(mask) %if the input data predates mpr version 2.3, mask
            is not in a cell array, and there can be only one mask so dump it into a
            cell array here
                mpr.mask{sensorindex}=mask; clear mask;
            else
                mpr.mask{sensorindex}={mask}; clear mask;
            end
            %mpr.maskelementsize(sensorindex,:)=maskelementsize;clear
            maskelementsize;
            mpr.sensordistance{sensorindex}=sensordistance;clear
            sensordistance;
            for ii=1:numel(mpr.sensordistance{sensorindex})
                mpr.maskelementsize{sensorindex}(ii,:)= [mpr.wx(sensorindex)/size(mpr.ma

```

```

sk{sensorindex}{ii},1),mpr.wy/size(mpr.mask{sensorindex}{ii},2)]; %if we
diced the mask, maskelementsize is not what we saved it as... HACK
KLUDGE
    end
    if exist('slope','var')
        mpr.slope(sensorindex)=slope;
    else
        mpr.slope(sensorindex)=.015; %vignetting slope, this should
be overwritten by the mask file as this is NOT a free parameter, but
some old mask files don't have slope stored ><
    end
case 'ASP'
    mpr.ASP_orientation{sensorindex}=ASP_orientation;
    mpr.ASP_alpha{sensorindex}=ASP_alpha;
    mpr.ASP_beta{sensorindex}=ASP_beta;
    mpr.ASP_m{sensorindex}=ASP_m;
otherwise
    error('unknown sensor category')
end
else
mpr.sensor_category='masked'; %before we added this var, all sensors
were of the masked type; add that datum and proceed.
if iscell(mask) %if the input data predates mpr version 2.3, mask is not
in a cell array, and there can be only one mask so dump it into a cell
array here
    mpr.mask{sensorindex}=mask; clear mask;
else
    mpr.mask{sensorindex}={mask}; clear mask;
end
mpr.sensordistance{sensorindex}=sensordistance;clear sensordistance;
for ii=1:numel(mpr.sensordistance{sensorindex})

    mpr.maskelementsize{sensorindex}(ii,:)= [mpr.wx(sensorindex)/size(mpr.ma
sk{sensorindex}{ii},1),mpr.wy/size(mpr.mask{sensorindex}{ii},2)]; %if we
diced the mask, maskelementsize is not what we saved it as... HACK
KLUDGE
    end
    if exist('slope','var')
        mpr.slope(sensorindex)=slope;
    else
        mpr.slope(sensorindex)=.015; %vignetting slope, this should be
overwritten by the mask file as this is NOT a free parameter, but some
old mask files don't have slope stored ><
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% sensorindex=2; % this is a counter increment for each sensor. it's like a
big unrolled loop, with a bunch of assignments for each sensor
% (no second sensor)

```

```
% %%%%%%%%%%  
end
```

whole_enchilada.m

```
% This is a series of scripts collated into a series that can run the  
% entire ILM simulation workflow when all run together  
% (the "whole enchilada"). They are organized into a series of matlab cells  
% that can be run individually (though not fully independently) to explore  
% changing certain parameters without re-running the entire script.  
%  
% Michael Henninger  
%  
% Last edit 5-26-2017 for comment updates, deletion of commented-out lines,  
% and whitespace cleanup  
  
%% initialize parameters  
enchilada_version='3.1.0';  
global sensor_orientations  
sensor_orientations=struct('zhat',1,'neg_zhat',2,'xhat',3,'neg_xhat',4);  
  
clusinfo=getCurrentCluster();  
if isempty(clusinfo) || strcmp(clusinfo.Type,'Local') %if this is being  
run locally, load the paths defined in matlab startup. clusinfo is empty  
if being run interactively or will have cluster type 'Local' if batch'ed  
on a local machine. The short circuit on the || prevents error on strcmp  
global photonsimpath  
global cspath  
global braincamdatapath  
else %if this is not being run locally ie, it is being run on Amazon EC,  
load the appropriate path  
photonsimpath='/shared/imported/photon_simulations';  
cspath='/shared/imported/CS/';  
braincamdatapath='/shared/persisted/results';  
end  
clear A; %if no clear A, out of memory when we do load A.  
clear nneurons; %must gen this carefully to avoid being out of sync, when  
adjusting number of neurons present  
  
%create_mpr=false; %create the mpr structure from script? if false you  
must load mpr from a file  
sprinkle_neurons=true; %run the segment of code that randomly places all  
the neurons?  
doctor_neurons=false; %alter number/location of specific neurons without  
completely redoing them with sprinkle_neurons. this will set  
gen_individual_images to 'all'  
gen_individual_images='all'; %run the segment of code that generates the  
sensor image caused by each individual neuron. if sprinkle_neurons is
```

```

true this must be 'all'. valid values are 'all' 'some' 'none'
gen_spike_distribution=true; %run the segment of code that generates
    spike times on each neuron
gen_sensor_images=true; %run the segment of code that generates the
    sensor image at each timestep due to all neurons and their spiking
gen_object_spaces=true; %run the segment of code that generates the 3d
    distribution of light power densities as simulated
gen_image_spaces=true; %run the segment of code that generates the 3d
    distribution of light power densities as reconstructed. also saves
    everything
solve_by_neuron=true; %rather than solve for each voxel, lump all the
    relevant voxels together and solve for each neuron. Lumping is done by
    matrix B_jun
densityset=[];%[1e4,3e4,5e4,7e4,1e5];
metaloop=1;
bigloop=1;
while bigloop<=20
    fprintf('setting parameters\n')

    if bigloop <= 20

mpr=mpr_param({'_quasi_1p12x1p12um_pixel_2p6um_mask_shift0_80x200_by_4um
    _200x200x100_sd5p0_OF50pct_NI1p4_40m_1'});
        if metaloop==1
            mpr.desired_neuron_density=1e5;
        else
            mpr.desired_neuron_density=5e4;
        end
        mpr.neuronbox=[-.07 .07 -.095 .095 .005 .095];
        mpr.deltaF_overF=.19;
        mpr.bitsPerPixel=10;
        mpr.fluencescale=30e3; %W/m^2 (divide by 1000 to get mW/mm^2,
divide by 10 go get mW/cm^2)
        mpr.background=false;
        mpr.shotnoise=true;
        mpr.sensornoise=true;
        mpr.nfgphotons=4e7;
        mpr.neuron_diameter=5;
        mpr.neuron_prefix='uniform';
        mpr.neuron_basename='y2000z2000at532nm_40m.mat';
        mpr.concentration=8*3e-5; %concentration of GCaMP6 in M
        mpr.QY=.59; %quantum yield at calcium saturation
        mpr.dynamic_range=11.24; %calcium saturated GCaMP is
mpr.dynamic_range brighter than baseline
        mpr.extinction=6.19e6; %extinction coefficient **in M^-1*m^-1**
        mpr.hnu=4.27e-19; %energy per excitation photon in joules (4.27e-
19 is for 465nm light)
        sprinkle_neurons=true;
        gen_spike_distribution=true;

```

```

    solve_by_neuron=true;
    mpr.position_error_type='systematic'; %are the position errors of
neurons' locations random or systematic? if systematic, use the below
line for direction(s) (ignored for random)
    mpr.position_error_magnitude=0; % in mm
    mpr.position_error_direction=[0,0,1]; %normalized direction
vector for directional errors. either one 3 element row vector or an
nneurons x 3 element set of row vectors for each neuron's displacement
    elseif bigloop <= 40

mpr=mpr_param_timeseries_2014({'_quasi_0p5x0p5um_pixel_2p6um_mask_shift0
_80x200_by_4um_200x200x100_sd5p0_OF50pct_NI1p4_40m_1'});
    if metaloop==1
        mpr.desired_neuron_density=1e5;
    else
        mpr.desired_neuron_density=5e4;
    end
    mpr.neuronbox=[-.07 .07 -.095 .095 .005 .095];
    mpr.deltaF_overF=.19;
    mpr.bitsPerPixel=10;
    mpr.fluencescale=30e3; %W/m^2 (divide by 1000 to get mW/mm^2,
divide by 10 go get mW/cm^2)
    mpr.background=false;
    mpr.shotnoise=false;
    mpr.sensornoise=false;
    mpr.nfgphotons=4e7;
    mpr.neuron_diameter=5;
    mpr.neuron_prefix='uniform';
    mpr.neuron_basename='y2000z2000at532nm_40m.mat';
    mpr.concentration=8*3e-5; %concentration of GCaMP6 in M
    mpr.QY=.59; %quantum yield at calcium saturation
    mpr.dynamic_range=11.24; %calcium saturated GCaMP is
mpr.dynamic_range brighter than baseline
    mpr.extinction=6.19e6; %extinction coefficient **in M^-1*m^-1**
    mpr.hnu=4.27e-19; %energy per excitation photon in joules (4.27e-
19 is for 465nm light)
    sprinkle_neurons=true;
    gen_spike_distribution=true;
    solve_by_neuron=true;
    mpr.position_error_type='systematic'; %are the position errors of
neurons' locations random or systematic? if systematic, use the below
line for direction(s) (ignored for random)
    mpr.position_error_magnitude=0; % in mm
    mpr.position_error_direction=[0,0,1]; %normalized direction
vector for directional errors. either one 3 element row vector or an
nneurons x 3 element set of row vectors for each neuron's displacement
    elseif bigloop <= 60

mpr=mpr_param_timeseries_2014({'_quasi_0p5x0p5um_pixel_2p6um_mask_shift0

```

```

_80x200_by_4um_200x200x100_sd5p0_OF50pct_NI1p4_40m_1'}));
    if metaloop==1
        mpr.desired_neuron_density=1e5;
    else
        mpr.desired_neuron_density=5e4;
    end
    mpr.neuronbox=[-.07 .07 -.095 .095 .005 .095];
    mpr.deltaF_overF=.19;
    mpr.bitsPerPixel=10;
    mpr.fluencescale=30e3; %W/m^2 (divide by 1000 to get mW/mm^2,
divide by 10 go get mW/cm^2)
    mpr.background=false;
    mpr.shotnoise=true;
    mpr.sensornoise=true;
    mpr.nfgphotons=4e7;
    mpr.neuron_diameter=5;
    mpr.neuron_prefix='uniform';
    mpr.neuron_basename='y2000z2000at532nm_40m.mat';
    mpr.concentration=8*3e-5; %concentration of GCaMP6 in M
    mpr.QY=.59; %quantum yield at calcium saturation
    mpr.dynamic_range=11.24; %calcium saturated GCaMP is
mpr.dynamic_range brighter than baseline
    mpr.extinction=6.19e6; %extinction coefficient **in M^-1*m^-1**
    mpr.hnu=4.27e-19; %energy per excitation photon in joules (4.27e-
19 is for 465nm light)
    sprinkle_neurons=true;
    gen_spike_distribution=true;
    solve_by_neuron=true;
    mpr.position_error_type='random'; %are the position errors of
neurons' locations random or systematic? if systematic, use the below
line for direction(s) (ignored for random)
    mpr.position_error_magnitude=.0025; % in mm
    mpr.position_error_direction=[0,0,1]; %normalized direction
vector for directional errors. either one 3 element row vector or an
nneurons x 3 element set of row vectors for each neuron's displacement
    elseif bigloop <= 80

mpr=mpr_param_timeseries_2014({'_quasi_0p5x0p5um_pixel_2p6um_mask_shift0
_80x200_by_4um_200x200x100_sd5p0_OF50pct_NI1p4_40m_1'}));
    if metaloop==1
        mpr.desired_neuron_density=1e5;
    else
        mpr.desired_neuron_density=5e4;
    end
    mpr.neuronbox=[-.07 .07 -.095 .095 .005 .095];
    mpr.deltaF_overF=.19;
    mpr.bitsPerPixel=10;
    mpr.fluencescale=30e3; %W/m^2 (divide by 1000 to get mW/mm^2,
divide by 10 go get mW/cm^2)

```

```

mpr.background=false;
mpr.shotnoise=true;
mpr.sensornoise=true;
mpr.nfgphotons=4e7;
mpr.neuron_diameter=5;
mpr.neuron_prefix='uniform';
mpr.neuron_basename='y2000z2000at532nm_40m.mat';
mpr.concentration=8*3e-5; %concentration of GCaMP6 in M
mpr.QY=.59; %quantum yield at calcium saturation
mpr.dynamic_range=11.24; %calcium saturated GCaMP is
mpr.dynamic_range brighter than baseline
mpr.extinction=6.19e6; %extinction coefficient **in M^-1*m^-1**
mpr.hnu=4.27e-19; %energy per excitation photon in joules (4.27e-
19 is for 465nm light)
sprinkle_neurons=true;
gen_spike_distribution=true;
solve_by_neuron=true;
mpr.position_error_type='random'; %are the position errors of
neurons' locations random or systematic? if systematic, use the below
line for direction(s) (ignored for random)
mpr.position_error_magnitude=.004; % in mm
mpr.position_error_direction=[0,0,1]; %normalized direction
vector for directional errors. either one 3 element row vector or an
nneurons x 3 element set of row vectors for each neuron's displacement
end
sensorindex=1; %HAX FIXME MAGIC NUMBER this breaks the ability to use
multiple sensors simultaneously, which can be re-added if I call
mpr_param_timeseries AFTER setting the per-run params above, and let it
only add default values. Should do that anyway

mpr.CSphotondensityscale(sensorindex)=(mpr.ncsphotons(sensorindex)/(mpr.
csvoxel(sensorindex,1)*mpr.csvoxel(sensorindex,2)*mpr.csvoxel(sensorinde
x,3)))/(mpr.nfgphotons/(4/3*pi*(mpr.neuron_diameter/2)^3)); %same as
photondensityscale but scaled to the CS voxels rather than the simulated
neurons.

mpr.photons_second_baseline=1/mpr.hnu*mpr.extinction*mpr.QY*mpr.concentr
ation*(4/3*pi*(mpr.neuron_diameter/2/1e6)^3)/mpr.dynamic_range; %photons
per cell per second per W/m^2 of excitation light at F0: I_excitation /
(energy per excitation photon) * (extinction coefficient) * (quantum
yield) * (protein concentration) * (cell volume)/(dynamic range) (the
factor of 1e6 converts from um to m
mpr.conversion_factor=mpr.photons_second_baseline/mpr.nfgphotons;
%multiply this by excitation intensity, divide by frame rate, then
multiply by the simulated sensor image to get photons per pixel per
frame
nframes=round(mpr.duration*mpr.framerate);

nneurons=round(mpr.desired_neuron_density*(mpr.neuronbox(2)-

```

```

mpr.neuronbox(1))*(mpr.neuronbox(4)-mpr.neuronbox(3))*(mpr.neuronbox(6)-
mpr.neuronbox(5));

    %% generate all the neuron locations. do this here so we can
independently change their brightness after multipdwrap runs its magic.
    if sprinkle_neurons
        fprintf('placing neurons\n')
        tic

[outshiftvecx,outshiftvecy,outshiftvecz]=scatterneurons2(mpr.dsensors,mpr.
r.psensors,mpr.csvoxel,mpr.csvol,mpr.neuronsspacing,mpr.neuronbox,mpr.neu
ron_diameter,mpr.nensors,nneurons);
        toc
    end
    if doctor_neurons
        %change outshiftvec variables, potentially changing their size,
        %also change nneuron variable to match length of outshiftvec's
        %
        %       cull_right=outshiftvecx<0;
        %       outshiftvecx=outshiftvecx(cull_right);
        %       outshiftvecy=outshiftvecy(cull_right);
        %       outshiftvecz=outshiftvecz(cull_right);
        %
        %%%do all your monkeying with neurons above here, then make sure
        %%%you've updated nneurons
        outshiftvecx(97)=outshiftvecx(97)+.002*bigloop
        nneurons=numel(outshiftvecx);
        gen_individual_images='some'; % ignore above set of this
variable; we need to update and/or resize the variables set in g_i_i
        image_gen_list=[97];
    end

    %% now generate the sensor image from each neuron
    rot_mat=zeros(3,3,numel(fieldnames(sensor_orientations))+1);
    rot_mat(:,:,sensor_orientations.zhat)=eye(3); % the default
orientation
    rot_mat(:,:,sensor_orientations.neg_zhat)=diag([-1,1,-1]); %if the
sensor normal is facing in negative z
    rot_mat(:,:,sensor_orientations.xhat)=[0,0,-1;0,1,0;1,0,0]; %sensor
normal facing +x
    rot_mat(:,:,sensor_orientations.neg_xhat)=[0,0,1;0,1,0;-1,0,0];
%sensor normal facing -x
    sort_orders=zeros(numel(outshiftvecx),mpr.nensors);
    errorvects=zeros(numel(outshiftvecx),3);
    if strcmp(mpr.position_error_type,'random')
        randvects=rand(numel(outshiftvecx),3);
        nrms=sqrt(sum(randvects.^2,2));
        randvects=bsxfun(@A./B,A./B,randvects,nrms); %normalizes
randvects.
        errorvects=randvects*mpr.position_error_magnitude; %scales the

```

```

errors
    elseif strcmp(mpr.position_error_type,'systematic')
        if size(mpr.position_error_direction,1)==1
            for iipem=1:3 %spatial dimensions

errorvects(:,iipem)=mpr.position_error_direction(iipem)*mpr.position_err
or_magnitude;
            end
        else

errorvects=mpr.position_error_direction*mpr.position_error_magnitude;
            end

        else
            error(strcat('position error type ',mpr.position_error_type,' not
recognized'))
            end

errorvects(:,1)=round(errorvects(:,1)/mpr.neuronspacing)*mpr.neuronspaci
ng; %put the X (and Z) coordinates on the neuron simulation grid

errorvects(:,3)=round(errorvects(:,3)/mpr.neuronspacing)*mpr.neuronspaci
ng;

    moutshiftvecx=outshiftvecx+errorvects(:,1);
    moutshiftvecy=outshiftvecy+errorvects(:,2);
    moutshiftvecz=outshiftvecz+errorvects(:,3);

    if strcmp(gen_individual_images,'all')
        fprintf('generating individual neuron sensor images\n')
        vpvfg=cell(mpr.nsenors,1);
        for ii=1:mpr.nsenors

vpvfg{ii}=zeros(ceil(mpr.wx(ii)/mpr.px(ii)),ceil(mpr.wy(ii)/mpr.py(ii)),
numel(outshiftvecx));
            end
            tic
            for ii=1:mpr.nsenors %loop through this once per sensor

this_outshiftvec=proper_transform(moutshiftvecx,moutshiftvecy,moutshiftv
ecz,mpr.psenors(ii,:)','mpr.dsenors(ii),'passive');
                %
                for jj=1:numel(outshiftvecx) %switch each
neuron to the local coordinates for sensor ii
                    %
                    this_outshiftvec(jj,:)=rot_mat(:, :,mpr.dsenors(ii))*([outshiftvecx(jj)
;outshiftvecy(jj);outshiftvecz(jj)]-mpr.psenors(ii,:)'); %translate
based on sensor's location, then rotate based on its orientation
                %
                end
                [dummy,sort_orders(:,ii)]=sort(this_outshiftvec(:,3)); %to

```

minimize file loads, we will sort by z-depth in the frame of the sensor.
save the sort order (but not sorted matrix) to apply to unsorted list
(see just below)

```

switch mpr.sensor_category
case 'masked'

```

```

vpvfg{ii}(:, :, sort_orders(:, ii))=individual_image_generator(mpr.sensor_c
ategory,mpr.wx(ii),mpr.wy(ii),mpr.px(ii),mpr.py(ii),mpr.neuron_prefix,mpr
r.neuron_diameter,mpr.neuron_basename,squeeze(mpr.sensorpos(ii,:)),this_
outshiftvec(sort_orders(:, ii),1),this_outshiftvec(sort_orders(:, ii),2),t
his_outshiftvec(sort_orders(:, ii),3),mpr.sensordistance{ii},mpr.slope(ii
),mpr.mask{ii},mpr.maskelements{ii},mpr.NOuter{ii},mpr.NInner{ii});
case 'ASP'

```

```

vpvfg{ii}(:, :, sort_orders(:, ii))=individual_image_generator(mpr.sensor_c
ategory,mpr.wx(ii),mpr.wy(ii),mpr.px(ii),mpr.py(ii),mpr.neuron_prefix,mpr
r.neuron_diameter,mpr.neuron_basename,squeeze(mpr.sensorpos(ii,:)),this_
outshiftvec(sort_orders(:, ii),1),this_outshiftvec(sort_orders(:, ii),2),t
his_outshiftvec(sort_orders(:, ii),3),mpr.ASP_orientation{ii},mpr.ASP_alp
ha{ii},mpr.ASP_beta{ii},mpr.ASP_m{ii},mpr.NOuter{ii},mpr.NInner{ii});
end

```

```

end
toc

```

```

end

```

```

if strcmp(gen_individual_images,'some') %only generate a few updated
neurons; keep all the rest the same.

```

```

fprintf('generating individual neuron sensor images FOR SELECTED
NEURONS\n')

```

```

tic

```

```

for ii=1:mpr.nensors

```

```

this_outshiftvec=proper_transform(moutshiftvecx,moutshiftvecy,moutshiftv
ecz,mpr.psensors(ii,:),'mpr.dsensors(ii),'passive');

```

```

% for jj=1: numel(outshiftvecx) %switch each
neuron to the local coordinates for sensor ii

```

```

%

```

```

this_outshiftvec(jj,:)=rot_mat(:, :,mpr.dsensors(ii))*([outshiftvecx(jj)
;outshiftvecy(jj);outshiftvecz(jj)]-mpr.psensors(ii,:)); %translate
based on sensor's location, then rotate based on its orientation

```

```

% end

```

```

[dummy,sort_orders(:, ii)]=sort(this_outshiftvec(:, 3)); %to
minimize file loads, we will sort by z-depth in the frame of the sensor.
save the sort order (but not sorted matrix) to apply to unsorted list
(see just below)

```

```

switch mpr.sensor_category
case 'masked'

```

```

vpvfg{ii}(:, :, sort_orders(image_gen_list, ii))=individual_image_generator
(mpr.sensor_category,mpr.wx(ii),mpr.wy(ii),mpr.px(ii),mpr.py(ii),mpr.neu

```

```

ron_prefix,mpr.neuron_diameter,mpr.neuron_basename,mpr.sensorpos(ii,:),t
his_outshiftvec(sort_orders(image_gen_list,ii),1),this_outshiftvec(sort_
orders(image_gen_list,ii),2),this_outshiftvec(sort_orders(image_gen_list
,ii),3),mpr.sensordistance{ii},mpr.slope(ii),mpr.mask{ii},mpr.maskelemen
tsize{ii},mpr.NOuter{ii},mpr.NInner{ii});
        case 'ASP'

vpvfg{ii}(:,:,sort_orders(image_gen_list,ii))=individual_image_generator
(mpr.sensor_category,mpr.wx(ii),mpr.wy(ii),mpr.px(ii),mpr.py(ii),mpr.neu
ron_prefix,mpr.neuron_diameter,mpr.neuron_basename,mpr.sensorpos(ii,:),t
his_outshiftvec(sort_orders(image_gen_list,ii),1),this_outshiftvec(sort_
orders(image_gen_list,ii),2),this_outshiftvec(sort_orders(image_gen_list
,ii),3),mpr.ASP_orientation{ii},mpr.ASP_alpha(ii),mpr.ASP_beta{ii},mpr.A
SP_m{ii},mpr.NOuter{ii},mpr.NInner{ii});
        end
    end
    toc

end
%% generate spike distribution
if gen_spike_distribution
    fprintf('generating spike distribution\n')
    spiketrace=zeros(mpr.duration*1000,nneurons,'single');
    spikeraster=zeros(mpr.duration*1000,nneurons,'single');
    tic
    spikeraster=single(rand(size(spikeraster))<(mpr.spikerate/1000));
%this will generate a poisson distribution of spikes. breaks when
spikerate approaches 1000hz...
    spikeraster(1:100,:)=0; % the risetime is 45 ms to peak, so if
first peak occurs at 100ms, the first 55ms are silent, baseline data.
    for iii=1:size(outshiftvecx,1)

spiketrace(:,iii)=conv(spikeraster(:,iii),mpr.spikeshape'*mpr.deltaF_ove
rF,'same');

        lightEDvoxel=[round((moutshiftvecx(iii)*1000-
mpr.lightingvol(1))/mpr.lightingvoxel(1)+1),round((moutshiftvecy(iii)*10
00-
mpr.lightingvol(3))/mpr.lightingvoxel(2)+1),round((moutshiftvecz(iii)*10
00-mpr.lightingvol(5))/mpr.lightingvoxel(3)+1)];

        if (lightEDvoxel(1)>size(mpr.lightingarray,1)) ||
(lightEDvoxel(2)>size(mpr.lightingarray,2)) ||
(lightEDvoxel(3)>size(mpr.lightingarray,3)) || prod(lightEDvoxel)<=0 %if
the neuron is located outside of the lighting array, set its brightness
to zero
            spiketrace(:,iii)=0;
            warning('at least one neuron lay outside the
lightingarray volume')

```

```

else

spiketrace(:,iii)=(spiketrace(:,iii)+mpr.baseline_fluorescence)*mpr.conv
ersion_factor*mpr.fluencescale/mpr.framerate*mpr.lightingarray(lightEDvo
xel(1),lightEDvoxel(2),lightEDvoxel(3));
    end
end

toc
end

%% generate actual sensor images caused by all neurons as they spike
if gen_sensor_images
    fprintf('generating sensor images\n')
    tic
    background_shot_noise{1:mpr.nensors}=[];
    background_frame{1:mpr.nensors}=[];
    frame{1:mpr.nensors}=[];
    frame_shot_noise{1:mpr.nensors}=[];
    for sensor_iteration=1:mpr.nensors

frame{sensor_iteration}=zeros(ceil(mpr.wx(sensor_iteration)/mpr.px(senso
r_iteration)),ceil(mpr.wy(sensor_iteration)/mpr.py(sensor_iteration)),nf
rames,'single');
        if mpr.shotnoise %if shotnoise is being used, we want to save
the shot-noisy version of each frame separately

frame_shot_noise{sensor_iteration}=zeros(ceil(mpr.wx(sensor_iteration)/m
pr.px(sensor_iteration)),ceil(mpr.wy(sensor_iteration)/mpr.py(sensor_ite
ration)),nframes,'single');

            if mpr.background

background_shot_noise{sensor_iteration}=zeros(ceil(mpr.wx(sensor_iterati
on)/mpr.px(sensor_iteration)),ceil(mpr.wy(sensor_iteration)/mpr.py(senso
r_iteration)),nframes,'single');
                end
            end
            if mpr.background

background_frame{sensor_iteration}=zeros(ceil(mpr.wx(sensor_iteration)/m
pr.px(sensor_iteration)),ceil(mpr.wy(sensor_iteration)/mpr.py(sensor_ite
ration)),'single'); %the mean background is constant across frames
unless there is dynamic lighting, which is not implemented

                end
                frametimes=round((1:nframes)/mpr.framerate*1000);

                %%I did not do a final check on this as I'm not using it

```

```

%%currently
if mpr.background==true

    [vox_map_x, vox_map_y,
vox_map_z]=ndgrid(mpr.csvol(sensor_iteration,1):mpr.csvoxel(sensor_ite
ration,1):mpr.csvol(sensor_iteration,2),mpr.csvol(sensor_iteration,3):mpr.
csvoxel(sensor_iteration,2):mpr.csvol(sensor_iteration,4),mpr.csvol(sens
or_iteration,5):mpr.csvoxel(sensor_iteration,3):mpr.csvol(sensor_iterati
on,6));

    vox_map_x=single(reshape(vox_map_x,[],1));
    vox_map_y=single(reshape(vox_map_y,[],1));
    vox_map_z=single(reshape(vox_map_z,[],1));

    %
testvarx=zeros(numel(vox_map_x),1,'single');
    %
testvarz=zeros(numel(vox_map_x),1,'single');

vox_loc_in_sensor_frame=proper_transform(vox_map_x,vox_map_y,vox_map_z,1
000*mpr.psensors(sensor_iteration,:),'mpr.dsensors(sensor_iteration),'ac
tive'); %the 1000 converts mm to um
    %
    % for quickloop=1:numel(vox_map_x)
    %
    lighting_voxel=[round((vox_loc_in_sensor_frame(1)-
mpr.lightingvol(1))/mpr.lightingvoxel(1)),round((vox_loc_in_sensor_frame
(2)-
mpr.lightingvol(3))/mpr.lightingvoxel(2)),round((vox_loc_in_sensor_frame
(3)-mpr.lightingvol(5))/mpr.lightingvoxel(3))]+1;
    %
    % if (lighting_voxel(1) > 0 &&
lighting_voxel(1) <= size(mpr.lightingarray,1) && (lighting_voxel(2) >
0 && lighting_voxel(2) <= size(mpr.lightingarray,2)) &&
(lighting_voxel(3) > 0 && lighting_voxel(3) <=
size(mpr.lightingarray,3))
    %
    background_illumination_by_voxel(quickloop)=mpr.lightingarray(lighting_
voxel(1),lighting_voxel(2),lighting_voxel(3));
    %
    % end
    %
    % end
    lighting_voxel=ceil([(vox_loc_in_sensor_frame(:,1)-
mpr.lightingvol(1))./mpr.lightingvoxel(1),(vox_loc_in_sensor_frame(:,2)-
mpr.lightingvol(3))./mpr.lightingvoxel(2),(vox_loc_in_sensor_frame(:,3)-
mpr.lightingvol(5))./mpr.lightingvoxel(3)]+1);
    lv_mask=lighting_voxel(:,1) > 0 & lighting_voxel(:,1) <=
size(mpr.lightingarray,1) & (lighting_voxel(:,2) > 0 &
lighting_voxel(:,2) <= size(mpr.lightingarray,2)) & lighting_voxel(:,3)
> 0 & lighting_voxel(:,3) <= size(mpr.lightingarray,3);

background_illumination_by_voxel=zeros(numel(vox_map_x),1,'single');

```

```

background_illumination_by_voxel(lv_mask)=mpr.lightingarray(sub2ind(size
(mpr.lightingarray),lighting_voxel(lv_mask,1),lighting_voxel(lv_mask,2),
lighting_voxel(lv_mask,3)))*mpr.conversion_factor*mpr.fluencescale/mpr.f
ramerate;

clear('vox_loc_in_sensor_frame','lighting_voxel','lv_mask');

        if
exist(fullfile(cspath, strcat('A',mpr.CS_basename{sensor_iteration},'_BAC
KGROUND')), 'file')==2

bga=load(strcat(cspath, 'A',mpr.CS_basename{sensor_iteration},'_BACKGROUN
D'),'A');
        else

bga=load(strcat(cspath, 'A',mpr.CS_basename{sensor_iteration}),'A');
        end

background_frame{sensor_iteration}=(reshape(bga.A*background_illuminatio
n_by_voxel,size(frame{sensor_iteration}(:, :,1)))/mpr.CSphotondensityscal
e(sensor_iteration)/mpr.fgbrightness);

        end

        for iii=1:nframes
            for jj=1:nneurons

frame{sensor_iteration}(:, :,iii)=frame{sensor_iteration}(:, :,iii)+(spike
trace(frametimes(iii),jj)*vpvfg{sensor_iteration}(:, :,jj));

            end
            if mpr.shotnoise

frame_shot_noise{sensor_iteration}(:, :,iii)=single(poissrnd(double(frame
{sensor_iteration}(:, :,iii)))); %an error in matlab (as of R2012B)
causes warnings--and possible math errors--when running poissrnd on
datatype single
                if mpr.background==true

background_shot_noise{sensor_iteration}(:, :,iii)=single(poissrnd(double(
background_frame{sensor_iteration}(:, :,iii))));
                    end
                end

            end

            %this spits out the right frame to use when doing
            %reconstruction, if doing reconstruction. needs to come

```

```

    %after all frames have been computed, and be done on each
    %sensor
    if gen_image_spaces
        if mpr.background
            if mpr.shotnoise

useThisFrame{sensor_iteration}=frame_shot_noise{sensor_iteration}+backgr
ound_shot_noise{sensor_iteration};
            else

useThisFrame{sensor_iteration}=bsxfun(@plus,frame{sensor_iteration},back
ground_frame{sensor_iteration});
            end
            else
                if mpr.shotnoise

useThisFrame{sensor_iteration}=frame_shot_noise{sensor_iteration};
            else

useThisFrame{sensor_iteration}=frame{sensor_iteration};
            end
            end
            %if we are using sensor noise, apply this here at the
            %end, to what we have already generated. This could be
            %calculated separately (there is read noise sewn in) but
can also be backed out from
            %data generated above
            if mpr.sensornoise

useThisFrame{sensor_iteration}=sensorNoise(useThisFrame{sensor_iteration
},mpr.electronsPerPhoton(sensor_iteration),mpr.readNoise(sensor_iteratio
n),mpr.bitsPerPixel(sensor_iteration),mpr.wellDepth(sensor_iteration));
            end
            end
            end
            toc
            clear bga
        end

    %% generate object spaces (3d voxel map of light emission
intensities)

    % a simple script to incorporate post-hoc knowledge of cell body
locations.
    % it just sets the columns of A that correspond to voxels without
cells to
    % zero. it requires the var objectspace which is generated by
    % plotimagespace.
    %objectbasis is used in other code segments and depends entirely on

```

```

the
    %variables below, and should be set according to them to make any
    %sense.
    fprintf('generating objectbasis\n')
    tic
    [objectbasis, object_vol,
neurNorm]=genobjectspace4(outshiftvecx,outshiftvecy,outshiftvecz,mpr.csv
oxel(1,:),mpr.csvol(1,:),mpr.psensors(1,:),mpr.neuron_diameter,mpr.neuro
n_prefix);% ,spiketrace(frametimes(framenumbers(ii)),:));
    toc
    if gen_object_spaces
        fprintf('generating object spaces\n')
        tic
        framenumbers=751:1:1300; %these are the sensor images we
reconstruct... can be a subset of all the sensor images generated by
timeseriesgenerator

objectspsaces=zeros(size(objectbasis,1),size(objectbasis,2),size(objectba
sis,3),numel(framenumbers),'single');

        for ii=1:numel(framenumbers)
            for jj=1:nneurons

objectspsaces(:,:,,ii)=objectspsaces(:,:,,ii)+(objectbasis(:,:,,jj)*spi
ketrace(frametimes(framenumbers(ii)),jj));
            end
        end
    toc
end
    %% incorporate post hoc knowledge and compute image space for each
frame, save the data
    if gen_image_spaces

        %pseudocode!
        %           get list of voxels from objectspace in um coordinates
        %           preallocate the masked A (narrower and taller than
real A
        %           for loop=1:#sensors
        %           proper_transform list of voxels into sensor's frame
        %           convert that list to csvoxel in sensor's frame
        %           for ii=1:#voxels (?? vectorize this?)
        %           add the corresponding column--or a column of zeros if
it lies outside csvol--to A_temp
        %           end
        %           attach A_temp to A.
        %           end
        %           make final frames by adding the backgrounds
        %           solve!!!

```

```

        fprintf('applying post hoc knowledge and generating image
space\n')
        tic
        imagespaces=zeros(size(objectsplaces),'single');

all_nonempty_voxels=find(objectsplaces(:,:,,1)>mpr.lightingcutoff);
    if solve_by_neuron

        B_jun=zeros(numel(all_nonempty_voxels),nneurons);
        for yy=1:nneurons
            temp=objectbasis(:,:,,yy);
            B_jun(:,yy)=temp(all_nonempty_voxels);
        end
    end

[nonzero_object_vox_x,nonzero_object_vox_y,nonzero_object_vox_z]=ind2sub
(size(objectsplaces(:,:,,1)),all_nonempty_voxels);

nonzero_object_x=object_vol(1)+(mpr.csvoxel(1)*(nonzero_object_vox_x-
1));

nonzero_object_y=object_vol(3)+(mpr.csvoxel(2)*(nonzero_object_vox_y-
1));

nonzero_object_z=object_vol(5)+(mpr.csvoxel(3)*(nonzero_object_vox_z-
1));
    vertical_step=zeros(mpr.nsenors,1);
    for ii=1:mpr.nsenors
        temp_A_obj=matfile(strcat(cspath,'A',mpr.CS_basename{ii}));
        vertical_step(ii)=size(temp_A_obj,'A',1);
    end
    clear temp_A_obj
    total_A_rows=sum(vertical_step);
    Amasked=zeros(total_A_rows,numel(nonzero_object_vox_x),'single');

    for masking_A_index=1:mpr.nsenors
        load(strcat(cspath,'A',mpr.CS_basename{masking_A_index}));
        %transformed loc is in sensor's frame, so csvol and csvoxel
are entirely relevant.

transformed_nonzero_loc=proper_transform(nonzero_object_x,nonzero_object
_y,nonzero_object_z,mpr.psenors(masking_A_index,:)*1000,mpr.dsenors(m
asking_A_index),'passive');
        transformed_vox=[ceil((transformed_nonzero_loc(:,1)-
mpr.csvol(masking_A_index,1))/mpr.csvoxel(masking_A_index,1)+1/2-1e-
8),ceil((transformed_nonzero_loc(:,2)-

```

```

mpr.csvol(masking_A_index,3))/mpr.csvoxel(masking_A_index,2)+1/2-1e-
8),ceil((transformed_nonzero_loc(:,3)-
mpr.csvol(masking_A_index,5))/mpr.csvoxel(masking_A_index,3)+1/2-1e-8)];

which_A_col=transformed_vox(:,1)+mpr.nx(masking_A_index)*(transformed_vo
x(:,2)-
1)+mpr.nx(masking_A_index)*mpr.ny(masking_A_index)*(transformed_vox(:,3)
-1); %note this has not yet checked for range overflow (ie, from a
neuron outside of this sensor's reconstruction volume)
    for vox_step=1:numel(nonzero_object_vox_x)
        %fprintf([num2str(vox_step),'\n'])
        is_valid=transformed_vox(vox_step,1)>0 &&
transformed_vox(vox_step,1)<=mpr.nx(masking_A_index) &&
transformed_vox(vox_step,2)>0 &&
transformed_vox(vox_step,2)<=mpr.ny(masking_A_index) &&
transformed_vox(vox_step,3)>0 &&
transformed_vox(vox_step,3)<=mpr.nz(masking_A_index); %if the voxel lies
outside reconstructed volume, is_valid is false and zeros are entered in
that part of A
        if is_valid
            if masking_A_index==1

Amasked(1:vertical_step(1),vox_step)=A(1:vertical_step(1),which_A_col(vo
x_step));

                else
                    Amasked(sum(vertical_step(1:(masking_A_index-
1)))+1:sum(vertical_step(1:masking_A_index)),vox_step)=A(1:vertical_step
(masking_A_index),which_A_col(vox_step));
                end
            else
                if masking_A_index==1

Amasked(1:vertical_step(1),vox_step)=zeros(vertical_step(1),1,'single');

                else
                    Amasked(sum(vertical_step(1:(masking_A_index-
1)))+1:sum(vertical_step(1:masking_A_index)),vox_step)=zeros(vertical_st
ep(masking_A_index),1,'single');
                end
            end
        end
    end

end

pixels_per_sensor=zeros(mpr.nsenors,1);
for ii=1:mpr.nsenors

pixels_per_sensor(ii)=ceil(mpr.wx(ii)/mpr.px(ii))*ceil(mpr.wy(ii)/mpr.py
(ii));

end

```

```

total_pixel_count=sum(pixels_per_sensor);
temp_shape=zeros(1,total_pixel_count);
finalframe=zeros(numel(framenumbers),total_pixel_count);
for ii=1:nframes;

temp_shape(1:pixels_per_sensor(1))=reshape(useThisFrame{1}(:, :, ii),1,[])
;
    for jj=2:mpr.nensors
        temp_shape(sum(pixels_per_sensor(1:(jj-
1))))+1:sum(pixels_per_sensor(1:jj))=reshape(useThisFrame{jj}(:, :, ii),1,
[]);
    end
    finalframe(ii,:)=temp_shape;
end
%%
for jj=1:numel(framenumbers)
    hmasking=framenumbers(jj);
    if jj==4
        hi=1;
    end

    if solve_by_neuron

hmasked=linsolve(Amasked*B_jun,finalframe(framenumbers(jj),:))'; %this
is where the magic happens.
        %hmasked=pinv(Amasked)*finalframe(framenumbers(jj),:));

imagespaces(:, :, :, jj)=genimagespace2(B_jun*hmasked,size(objectspaces(:, :,
:, 1)),find(objectspaces(:, :, :, 1)>mpr.lightingcutoff));
        else

hmasked=linsolve(Amasked,finalframe(framenumbers(jj),:))'; %this is
where the magic happens.
        %hmasked=pinv(Amasked)*finalframe(framenumbers(jj),:));

imagespaces(:, :, :, jj)=genimagespace2(hmasked,size(objectspaces(:, :, :, 1))
,find(objectspaces(:, :, :, 1)>mpr.lightingcutoff));
        end
    end
    %clear('hmasked','hfull','Amasked')
    imagespacetraces=zeros(numel(framenumbers),nneurons,'single');
    objectspacetraces=zeros(numel(framenumbers),nneurons,'single');

normedimagespacetraces=zeros(numel(framenumbers),nneurons,'single');

normedobjectspacetraces=zeros(numel(framenumbers),nneurons,'single');
    for ii=1:numel(framenumbers)

imagespacetraces(ii,:)=poll_voxels2(objectbasis,imagespaces(:, :, :, ii));

```

```

objectspacetraces(ii,:)=poll_voxels2(objectbasis,objectspaces(:,:,,ii))
;
    end
    for ii=1:numel(framenumbers)

normedimagespacetraces(ii,:)=imagespacetraces(ii,:)./imagespacetraces(1,
:);

normedobjectspacetraces(ii,:)=objectspacetraces(ii,:)./objectspacetraces
(1,:);
    end
    renormimage=zeros(size(imagespacetraces));
    renormobject=zeros(size(objectspacetraces));
    renormtrace=zeros(numel(frametimes),size(spiketrace,2));
    for ii=1:numel(outshiftvecz)
        renormimage(:,ii)=(imagespacetraces(:,ii)-
mean(imagespacetraces(:,ii)))/(max(imagespacetraces(:,ii))-
min(imagespacetraces(:,ii)));
        renormobject(:,ii)=(objectspacetraces(:,ii)-
mean(objectspacetraces(:,ii)))/(max(objectspacetraces(:,ii))-
min(objectspacetraces(:,ii)));
        renormtrace(:,ii)=(spiketrace(frametimes,ii)-
mean(spiketrace(frametimes,ii)))/(max(spiketrace(frametimes,ii)-
min(spiketrace(frametimes,ii))));
    end
    clear('A','photonPhaseTotal','ii')
    BCON=strcat(braincamdatapath,'2014\');
    if ~exist(BCON,'dir')
        mkdir(BCON)
    end
    next_file_number=numel(dir(strcat(BCON,'\*.mat')))+1;

save_these_vars={'frame_shot_noise','background_frame','background_shot_
noise','frame','framenumbers',
'frametimes','gen_image_spaces','gen_individual_images','gen_object_spac
es','gen_sensor_images','gen_spike_distribution','solve_by_neuron','imag
espaces','imagespacetraces','mpr','nframes','nneurons','normedimagespace
traces','normedobjectspacetraces','objectspaces','objectspacetraces','ou
tshiftvecx','outshiftvecy','outshiftvecz','moutshiftvecx','moutshiftvecy
','moutshiftvecz','spikeraster','spiketrace','sprinkle_neurons','doctor_
neurons','vpvfg','renormimage','renormtrace','renormobject','enchilada_v
ersion','solve_by_neuron','neurNorm'};

save(strcat(BCON,'\',num2str(next_file_number)),save_these_vars{:})
    toc
    end
    if bigloop==-1 && metaloop==1

```

```

        metaloop=2
        bigloop=1
    else
        bigloop=bigloop+1
    end
end
end

```

genimagespace2.m

```

function [ imagespace ] = genimagespace2(
    Hmasked,object_space_size,nonzero_indices )
%genimagespace takes a long vector H corresponding to all the non-zero voxels
in the
%image space, and reshapes them into the 3-D volume matching the dimensions
of the object space

imagespace=zeros(object_space_size,'single');
for ii=1:numel(nonzero_indices)
    imagespace(nonzero_indices(ii)) = Hmasked(ii);
end

end

```

genObjectSpace.m

```

function [ objectspace, local_object_volume, norm_each_neuron ] =
    genobjectspace4( outshiftvecx, outshiftvecy, outshiftvecz, localcsvoxel,
        localcsvol,localpsensor,neuron_diameter,neuron_prefix)
%genobjectspace generates the simulated light intensity of the object space.
the geometry done is quick and DIRTY.
%genobjectspace4 adds the norm_each_neuron return and the ability to
%calculate shells.
%introduced in enchilada version 2.7.2

%There is bad accuracy (though no grossly wrong results) in determining which
voxels are fully/partially/not enclosed by neurons
%(corners might be in/out of neuron but not detected as such)
%the fraction of voxel filled by neuron is a bad linear approximation.
%these are quantitative inaccuracies, not qualitative errors though.
if abs(localcsvoxel(1)-localcsvoxel(2))>1e-6 || abs(localcsvoxel(2)-
    localcsvoxel(3))>1e-6
    error('reconstruction volume voxels are not cubic, incompatible with
    genobjectspace4')
end
cs_ref_voxel=localcsvol([1,3,5])-localpsensor*1000;
x_voxel_offset=rem(cs_ref_voxel(1),localcsvoxel(1));if
    x_voxel_offset<0;x_voxel_offset=x_voxel_offset+localcsvoxel(1);end;
%voxel edge alignment can be in different places (notably, is 0 the
center of a voxel or the edge?) so find an offset to get on the right
frame

```

```

y_voxel_offset=rem(cs_ref_voxel(2),localcsvoxel(2));if
    y_voxel_offset<0;y_voxel_offset=y_voxel_offset+localcsvoxel(2);end;
    %this tells us how far the center of a voxel is shifted from 0
z_voxel_offset=rem(cs_ref_voxel(3),localcsvoxel(3));if
    z_voxel_offset<0;z_voxel_offset=z_voxel_offset+localcsvoxel(3);end;
xlb_temp=min(outshiftvecx)*1000-neuron_diameter/2; %the x coordinate of the
    left edge of the leftmost neuron
xlb=ceil(xlb_temp/localcsvoxel(1))*localcsvoxel(1)+x_voxel_offset;while xlb-
    localcsvoxel(1)/2>xlb_temp;xlb=xlb-localcsvoxel(1);end; %put the x lower
    bound on the csvoxel grid, and if it doesn't reach all the way after
    putting on grid, shift it one grid step left. this is coordinate for
    CENTER of voxel
ylb_temp=min(outshiftvecy)*1000-neuron_diameter/2; %the y coordinate of the
    left edge of the bottommost neuron
ylb=ceil(ylb_temp/localcsvoxel(2))*localcsvoxel(2)+y_voxel_offset;while ylb-
    localcsvoxel(2)/2>ylb_temp;ylb=ylb-localcsvoxel(2);end; %put the y lower
    bound on the csvoxel grid, and if it doesn't reach all the way after
    putting on grid, shift it one grid step down
zlb_temp=min(outshiftvecz)*1000-neuron_diameter/2; %the x coordinate of the
    left edge of the leftmost neuron
zlb=ceil(zlb_temp/localcsvoxel(3))*localcsvoxel(3)+z_voxel_offset;while zlb-
    localcsvoxel(3)/2>zlb_temp;zlb=zlb-localcsvoxel(3);end; %put the z lower
    bound on the csvoxel grid, and if it doesn't reach all the way after
    putting on grid, shift it one grid step forward
xub_temp=max(outshiftvecx)*1000+neuron_diameter/2; %the x coordinate of the
    left edge of the leftmost neuron
xub=floor(xub_temp/localcsvoxel(1))*localcsvoxel(1)-x_voxel_offset;while
    xub+localcsvoxel(1)/2<xub_temp;xub=xub+localcsvoxel(1);end;
yub_temp=max(outshiftvecy)*1000+neuron_diameter/2; %the y coordinate of the
    left edge of the bottommost neuron
yub=floor(yub_temp/localcsvoxel(2))*localcsvoxel(2)-y_voxel_offset;while
    yub+localcsvoxel(2)/2<yub_temp;yub=yub+localcsvoxel(2);end;
zub_temp=max(outshiftvecz)*1000+neuron_diameter/2; %the x coordinate of the
    left edge of the leftmost neuron
zub=floor(zub_temp/localcsvoxel(3))*localcsvoxel(3)-z_voxel_offset;while
    zub+localcsvoxel(3)/2<zub_temp;zub=zub+localcsvoxel(3);end;
local_object_volume=[xlb, xub, ylb, yub, zlb, zub];
onx=round((xub-xlb)/localcsvoxel(1))+1;
ony=round((yub-ylb)/localcsvoxel(2))+1;
onz=round((zub-zlb)/localcsvoxel(3))+1;
objectspace=zeros(onx,ony,onz,size(outshiftvecz,1),'single');
norm_each_neuron=zeros(size(outshiftvecz,1),'single');
for qq=1:size(outshiftvecz)
    currx=(outshiftvecx(qq)*1000-xlb)/localcsvoxel(1); %the location of
        neurons, in units of voxel size, measured from the center of the
        negativemost corner voxel of the object volume
    curry=(outshiftvecy(qq)*1000-ylb)/localcsvoxel(2);
    currz=(outshiftvecz(qq)*1000-zlb)/localcsvoxel(3);
    rinvoxels=neuron_diameter/2/localcsvoxel(1); %radius of neuron in terms

```

of voxels

%I could probably vectorize this loop with bsxfun? arrayfun? something.

switch neuron_prefix

case 'uniform'

for ii=1:onx

for jj=1:ony

for kk=1:onz

voxelstocell=sqrt(((ii-1)-currx)^2+((jj-1)-
curry)^2+((kk-1)-currz)^2); %distance from the center of the voxel to
the center of the cell

if voxelstocell+.5 < rinvoxels %if the distance to
the far side of the voxel (voxelstocell+1/2 of a voxel...1/2 a voxel the
distance from center to side of voxel) is less than the radius, the
voxel is basically fully enclosed by cell

objectspace(ii,jj,kk,qq)=1;

norm_each_neuron(qq)=norm_each_neuron(qq)+1;

elseif voxelstocell-.5 < rinvoxels %if the near side
of the voxel is inside the radius of the neuron, do a BAD linear
approximation of volume fraction

objectspace(ii,jj,kk,qq)=(rinvoxels-
voxelstocell+.5);

norm_each_neuron(qq)=norm_each_neuron(qq)+(rinvoxels-voxelstocell+.5);

end

end

end

end

case 'Shell'

for ii=1:onx

for jj=1:ony

for kk=1:onz

V=[(ii-1)-currx,(jj-1)-curry,(kk-1)-currz]; %vector
from center of shell to center of voxel

eintersects=findEdgeIntersects(V,rinvoxels);

if numel(eintersects)>0 %if we intersect within the
voxels, figure out the area of the intersection

if numel(eintersects)<3 %it should intersect at
enough points to make a polygon if it intersects at all!

error('wtf? strange number of intersections')

end

aEnc=computeConvexArea(eintersects,V);

objectspace(ii,jj,kk,qq)=aEnc;

norm_each_neuron(qq)=norm_each_neuron(qq)+aEnc;

end

end

end

end

```

        otherwise
            error(strcat('error identifying source shape; could not generate
objectspace for prefix ',neuron_prefix))
        end
    end
end
end

```

computeConvexArea.m

```

function [ zeArea ] = computeConvexArea( zePoints, V )
    function bt=aequal(a,b)
        bt= (abs(a-b)/(abs(a)+abs(b))<1e-3);
    end
    function [sortOrd]=sortCCW(planarPoints)
        meanx=mean(planarPoints(:,1));
        meany=mean(planarPoints(:,2));
        tempscratch=cart2pol(planarPoints(:,1)-meanx,planarPoints(:,2)-
        meany);
        [~,sortOrd]=sort(tempscratch(:,1));
    end
    function [twoDArea]=twoDA(SPP)
        npts=size(SPP,1);
        twoDArea=0;
        for ii=1:(npts-1) %need to add wraparound to first point after this
        loop
            twoDArea=twoDArea+SPP(ii,1)*SPP(ii+1,2)-SPP(ii+1,1)*SPP(ii,2);
        end
        twoDArea=twoDArea+SPP(npts,1)*SPP(1,2)-SPP(1,1)*SPP(npts,2);
        twoDArea=twoDArea/2; %I've been leaving out a factor of two above for
        convenience.

    end
    nearestToNormal=find(abs(V)==max(abs(V)));
    nearestToNormal=nearestToNormal(1); %if V's largest component is shared
    by more than one direction, just pick one as they are equivalent. can't
    let it be more than one.
    otherTwo=setxor(1:3,nearestToNormal);
    SO=sortCCW(zePoints(:,otherTwo));
    sortedPoints=zePoints(SO,:);
    projArea=twoDA(sortedPoints(:,otherTwo));
    zeArea=projArea*sqrt(V*V')/V(nearestToNormal);
end

```

findEdgeIntersects.m

```

function [ goodintersections ] = findEdgeIntersects( V, r )
%where does a plane perpendicular to V at a distance rinvexels from origin
%intersect the lines coinciding with edges of a unit cube?
% Detailed explanation goes here
LPintersect=@(a,r,Vx,Vy,Vz,x,y) (r^2*a-x*Vx-y*Vy)/Vz; %line-plane

```

intersection, determining the one missing coordinate (lines are voxel edges, parallel with axes so only one unknown) NOTE SWAP YOUR AXES LABELS AS NECESSARY

```

voxelstocell=sqrt(V*V'); %distance from the center of the voxel to the
  center of the cell, in voxel edge lengths
a=voxelstocell/r; %rinvoxels*scaleA=voxelstocell; a scale factor
%Vhat=V/(r*a); %unit vector pointing along V, perpto the plane
  approximating surface of shell
intersections=ones(12,3)*1e6; %this will record the intersections of the
  tangent plane with all the lines coincident with the voxel's edges. if
  something reads 1e6 it hasn't been set and we shouldn't be using it
intersections([4,5,8,12],1)=V(1)-.5; %doing this relative to sphere
  center
intersections([2,6,7,10],1)=V(1)+.5;
intersections([1,5,6,9],2)=V(2)-.5;
intersections([3,7,8,11],2)=V(2)+.5;
intersections(1:4,3)=V(3)-.5;
intersections(9:12,3)=V(3)+.5;
fl=[2,3,1 %first two columns are which indices are fixed on each
  edge, third column is the one that varies
  1,3,2
  2,3,1
  1,3,2
  1,2,3
  1,2,3
  1,2,3
  1,2,3
  1,2,3
  2,3,1
  1,3,2
  2,3,1
  1,3,2];
for ii=1:12 %once for each edge

  mv=LPintersect(a,r,V(fl(ii,1)),V(fl(ii,2)),V(fl(ii,3)),intersections(ii,
    fl(ii,1)),intersections(ii,fl(ii,2)));
  if ~(isnan(mv) || isinf(mv))
    intersections(ii,fl(ii,3))=mv;
  end
end

%translate back into voxel-centered coordinates for convenience in next
%calculation
for ii=1:12
  intersections(ii,:)=intersections(ii,:)-V;
end
%now do a quick cull of "outside of voxel" intersections with voxel
%edge lines
goodpoints=~(max(abs(intersections),[],2)>.5);

```

```

    goodintersections=intersections(goodpoints,:);
end

individual_image_generator.m
function [ vpvfg ] =
    individual_image_generator(sensor_category,wx,wy,px,py,neuron_prefix,neu
        ron_diameter,neuron_basename,sensorpos,outshiftvecx,outshiftvecy,outshif
            tvecz, varargin)
%this is a wrapper for vpvmaskgen and vpvASPgen
global photonsimpath
switch sensor_category
    case 'masked'
        sensordistance=varargin{1};
        slope=varargin{2};
        mask=varargin{3};
        maskelementsize=varargin{4};
        NOuter=varargin{5};
        NInner=varargin{6};
    case 'ASP'
        ASP_orientation=varargin{1};
        ASP_alpha=varargin{2};
        ASP_beta=varargin{3};
        ASP_m=varargin{4};
        NOuter=varargin{5};
        NInner=varargin{6};

end

nentries=numel(outshiftvecx);
vpvfg=zeros(ceil(wx/px),ceil(wy/py),nentries);
loadeddist=-1e6; %this is not a valid loaded distance; it's just sure to
    trigger loading a distance on the first iteration
for neurloop=1:nentries;
    if abs(loadeddist-outshiftvecz(neurloop))>1e-5 % if it has not
        successfully loaded the file for neurons at distance outshiftvecz

        filenm=strcat(photonsimpath,neuron_prefix,'d',num2str(neuron_diameter),
            'x',num2str(round(outshiftvecz(neurloop)*1000)),neuron_basename);
        if exist(filenm,'file')
            load(filenm,'photonPhaseTotal')
            loadeddist=outshiftvecz(neurloop);
        else
%            if loadeddist>0 %if loadeddist is positive and we can't find a
                file, we might need to simulate more neuron distances rather than fail
                    quietly. On the other hand, the neuron could be really far away and
                        maybe we do want to ignore it so just warn
                            warning(strcat('could not find a monte carlo file
                                named',filenm))
%                            end
end
end

```

```

        continue %if we can't load up photons from a given neuron, skip
        generating the sensor image, which leaves that neuron's entry in vpvfg
        as totally dark (all zeros), which is a good way to handle this
        condition
    end
end

switch sensor_category
case 'masked'

    vpvfg(:, :, neurloop)=vpvmaskgen3(px,py,wx,wy,sensordistance,slope,outshi
ftvecx(neurloop),outshiftvecy(neurloop),photonPhaseTotal,sensorpos,mask,
maskelementsizе,NOuter,NInner);
    case 'ASP'

        vpvfg(:, :, neurloop)=vpvASPgen(px,py,wx,wy,outshiftvecx(neurloop),outshi
ftvecy(neurloop),photonPhaseTotal,sensorpos,ASP_orientation,ASP_alpha,AS
P_beta,ASP_m,NOuter,NInner);
    end
end
end

```

poll_voxels2.m

```

function [ currentvalues ] = poll_voxels2(objbasis,theframe )
% POLL_VOXELS gets the intensity from the voxels cotaining any part of a
% given neuron

```

```

currentvalues=zeros(size(objbasis,4),1);
for ii=1:size(objbasis,4)

    currentvalues(ii)=sum(sum(sum((objbasis(:, :, :, ii)>0).*theframe)));
end

end

```

proper_transform.m

```

function [ output_points ] =
    proper_transform(x_in,y_in,z_in,translation_vector,orientation,sense )
%x_in,y_in,z_in are column vectors of all the coordinates that need
transforming
%translation vector is translation component of transform
%orientation is the orientation of the new frame, which is one of 4 cardinal
directions in the x-z plane. corresponds to rotational component of
transform
%sense is either active or passive, depending on if the transform itself is
active or passive. passive for neurons, active for CS voxels w.r.t.
lightingarray, etc.

```

```

global sensor_orientations

```

```

rot_mat=zeros(3,3,numel(fieldnames(sensor_orientations))+1);
rot_mat(:,:,sensor_orientations.zhat)=eye(3); % the default orientation
rot_mat(:,:,sensor_orientations.neg_zhat)=diag([-1,1,-1]); %if the sensor
    normal is facing in negative z
rot_mat(:,:,sensor_orientations.xhat)=[0,0,-1;0,1,0;1,0,0]; %sensor normal
    facing +x
rot_mat(:,:,sensor_orientations.neg_xhat)=[0,0,1;0,1,0;-1,0,0]; %sensor
    normal facing -x

if strcmp(sense,'passive')
    [output_x,output_y,output_z]=arrayfun(@transform_one_passive,x_in,y_in,z_
_in);
else
    [output_x,output_y,output_z]=arrayfun(@transform_one_active,x_in,y_in,z_
_in);
end
output_points=[output_x,output_y,output_z];

%nested functions only down here
function
[one_out_x,one_out_y,one_out_z]=transform_one_passive(input_x,input_y,in
put_z)
    rolled_up=rot_mat(:,:,orientation)*([input_x;input_y;input_z]-
translation_vector);
    one_out_x=rolled_up(1);
    one_out_y=rolled_up(2);
    one_out_z=rolled_up(3);
end
function
[one_out_x,one_out_y,one_out_z]=transform_one_active(input_x,input_y,inp
ut_z)

    rolled_up=inv(rot_mat(:,:,orientation))*[input_x;input_y;input_z]+trans
lation_vector;
    one_out_x=rolled_up(1);
    one_out_y=rolled_up(2);
    one_out_z=rolled_up(3);
end
end

scatterneurons2.m
function [ outshiftvecx,outshiftvecy,outshiftvecz ] =
    scatterneurons2(dsensors,psensors,csvoxel,csvol,neuronspacing,neuronbox,
    neuron_diameter,nsensors,nneurons)
% scatters neurons in the object space, making sure they don't physically
% overlap each other. Also check to make sure the sensors
global sensor_orientations
outshiftvecx=zeros(nneurons,1);

```

```

outshiftvecy=zeros(nneurons,1);
outshiftvecz=zeros(nneurons,1);
if (any(dsensors==sensor_orientations.xhat |
    dsensors==sensor_orientations.neg_xhat) &&
    any(dsensors==sensor_orientations.zhat |
    dsensors==sensor_orientations.neg_zhat)) %if we have two orthogonal
    sensors
    for ii=1:nsensors
        if csvoxel(ii,1)~=csvoxel(ii,3)
            error('braincam:bad_csgrid_align','you have sensors with normals
in both x and z directions, but csvoxels that are not square in the xz
plane. This ruins csvoxel coregistration')
        end
    end
end
for ii=2:numel(dsensors)
    test_voxel=[csvol(1,1)+psensors(1,1),csvol(1,3)+psensors(1,2),csvol(1,5)
+psensors(1,3)];
    test_voxel_moved=proper_transform(test_voxel(1),test_voxel(2),test_voxel
(3),1000*psensors(ii,:),dsensors(ii),'passive');
    if rem(test_voxel_moved(1),test_voxel(1))>1e-6 ||
rem(test_voxel_moved(2),test_voxel(2))>1e-6 ||
rem(test_voxel_moved(3),test_voxel(3))>1e-6
        error('braincam:bad_csgrid_align',strcat('the csvoxels of sensor
#',num2str(ii),' do not align with the csvoxels of sensor #1'))
    end
end
sensor_grid_spacing_x=lcm(neuronspacing*1e7,csvoxel(1,1)/1e3*1e7)/1e7; %lcm
only works for integers hence the 1e7 scaling. putting sensors at a
multiple of this value ensures that both the positions of the csvoxels
and the positions of the neurons are compatible with all sensors, in the
x direction
sensor_grid_spacing_z=lcm(neuronspacing*1e7,csvoxel(1,3)/1e3*1e7)/1e7; %same
as above, for z
bsa_error_indices=[];
for ii=1:nsensors
    if dsensors(ii)==sensor_orientations.xhat &&
rem(psensors(ii,1),sensor_grid_spacing_x)>1e-6
        bsa_error_indices=[bsa_error_indices,ii];
    end
    if dsensors(ii)==sensor_orientations.zhat &&
rem(psensors(ii,3),sensor_grid_spacing_z)>1e-6
        bsa_error_indices=[bsa_error_indices,ii];
    end
end
if numel(bsa_error_indices)>0
    error('braincam:bad_sensor_align',strcat('sensor is not located on
sensor grid spacing. problem with sensor ',num2str(bsa_error_indices)))
end

```

```

for qq=1:nneurons
    hyp(1)=round((neuronbox(1)+(neuronbox(2)-
        neuronbox(1))*rand(1))/neuronspacing)*neuronspacing; %randomly place
        neurons in the neuronbox area (which sets bounds for simulated volume)
    hyp(2)=neuronbox(3)+(neuronbox(4)-neuronbox(3))*rand(1);
    hyp(3)=round((neuronbox(5)+(neuronbox(6)-
        neuronbox(5))*rand(1))/neuronspacing)*neuronspacing; %divide by neuron
        spacing to bring it into integer mm range, round, then mult. by neuron
        spacing to get it back to correct sub-mm range
    ticker=0;
    while any(((hyp(1)-outshiftvecx(1:qq)).^2+(hyp(2)-
        outshiftvecy(1:qq)).^2+(hyp(3)-
        outshiftvecz(1:qq)).^2)<(neuron_diameter/1000)^2) %if the newly placed
        randomly located neuron overlaps an existing neuron, junk it and try
        again. neuron_diameter is in microns (argh!)
        hyp(1)=round((neuronbox(1)+(neuronbox(2)-
        neuronbox(1))*rand(1))/neuronspacing)*neuronspacing;
        hyp(2)=neuronbox(3)+(neuronbox(4)-neuronbox(3))*rand(1);
        hyp(3)=round((neuronbox(5)+(neuronbox(6)-
        neuronbox(5))*rand(1))/neuronspacing)*neuronspacing;
        ticker=ticker+1;
        if ticker>1e5 %if we've tried like hell to add a new neuron and
        can't, warn (punting!) and just keep going
            warning('unable to sprinkle a neuron after 100,000
            tries...density likely too high')
            continue
        end
    end
    outshiftvecx(qq)=hyp(1);
    outshiftvecy(qq)=hyp(2);
    outshiftvecz(qq)=hyp(3);
end
end

```

sensorNoise.m

```

function
    [outframe]=sensorNoise(inframe,electronsPerPhoton,readNoise,bitsPerPixel
        ,wellDepth)
%a function to convert from photons/pixel to e-/pixel and add read noise and
%such. it takes the images of a single sensor (3D data, 2 space 1 time, call
    using sensorNoise(frame{1},...) or whatever) and converts.

    mesoframe=inframe*electronsPerPhoton;

    mesoframe=mesoframe+randn(size(mesoframe))*readNoise;
    mesoframe=mesoframe/(wellDepth/2^bitsPerPixel);
    mesoframe(mesoframe<=0)=1e-6; %if the gaussian assumption of the noise
    led to negative electrons, fix that to 0+'eps' (eps so that we don't get

```

```

    2^n+1 levels, just 2^n).
    outframe=min(ceil(mesoframe),2^bitsPerPixel); %our digitizer is one
    indexed. report either the value of the pixel measured in levels of the
    digitizer, or the max value if it's beyond saturation

end

vpvmaskgen.m
function [ vPixelVig ] = vpvmaskgen3( py,pz,wy,wz,sensordistance,slope, ...
    shiftvecy,shiftvecz,photonPhaseTotal,sensorpos,mask,maskelementsiz,NOuter,NInner )
% this file extracts the generation of vPixelVig from the prior, larger file
    (eg pdfunc)
%6-11-2013: made masks accept non-binary values (for partially opaque pixels)
%6-11-2013: allowed multiple masks per sensor

%vPixelVig is an array of the intensities on each pixel of the actual sensor.
    It takes the list of photons (their position and direction) hitting the
    pinholes
%(from photonPhaseTotal) and projects it through pinholes onto the sensor
    plane.

%note BIG CHANGE: prad is now in mm, NOT pixels, as prad is not required to
    be some pixel multiple now.
%% setup code
%these are constants, derived parameters from the ones passed, load file,
    etc.
X=1; %just an enumeration for convenience
Y=2;
Z=3;
VX=4; %these are the components of the normalized velocity
VY=5;
VZ=6;
WEIGHT=7;
near=1e-8; %since we're working with floats we need to be ready for photons
    that aren't exactly at the boundary.
vPixelVig=zeros(ceil(wy/py),ceil(wz/pz));

%% load and preprocess the photons
%load command taken out in favor of just passing photonPhaseTotal
%directly... load it in your wrapper code.
%load(filename,'photonPhaseTotal') %this loads the file with the simulated
    photons...it is used to get photonPhaseTotal.
photonPhaseTotal(:,Y)=photonPhaseTotal(:,Y)+shiftvecy-sensorpos(1);
photonPhaseTotal(:,Z)=photonPhaseTotal(:,Z)+shiftvecz-sensorpos(2);

%note sensorpos is a 2 element vector with [starty, startz] of the sensor
    (ie, top left corner). sensor width and height is wy and wz
%this doesn't need to match all the photons. if a photon is not on the

```

```

    sensor, it is ignored
TIRCutoff=asind(min(NInner)/NOuter); % angles greater than TIRCutoff are
    totally internally refracted at one of the boundaries. remove them.
thetadegOuter=real(acosd(-photonPhaseTotal(:,VX))); %this is the angle in
    degrees with respect to normal, matters for pixel vignetting
tempmask=(photonPhaseTotal(:,Y)>0) & (photonPhaseTotal(:,Y)<=wy) &
    (photonPhaseTotal(:,Z)>0) & (photonPhaseTotal(:,Z)<=wz) &
    (abs(photonPhaseTotal(:,X))<near) & (thetadegOuter<TIRCutoff);
photonPhaseTotal=photonPhaseTotal(tempmask,:);
clear tempmask
%now mask again, for stuff that hit pinholes rather than landed between them.
    This is gonna make this code REAL fast.
%square pinholes (prad is thus a side length of the square pinhole)
%tempmask=(abs(mod(photonPhaseTotal(:,Y),psep*py)-psep*py/2) < prad) &
    (abs(mod(photonPhaseTotal(:,Z),psep*pz)-psep*pz/2)<prad);
%this requires linear indexing to get all the elements out in a list, as
    normal indexing of two lists (one for column and one for row) returns a
    full matrix that is both huge and not what we want

%%REPLACED 6-11-2013. this code removed in favor of allowing partially
    opaque elements. will cause photon propagation to be slower, but more
    flexible.
%indx=sub2ind(size(mask),ceil(photonPhaseTotal(:,Y)/maskelements(1)),ceil(
    photonPhaseTotal(:,Z)/maskelements(2)));
% tempmask=cast(mask(indx),'logical'); %note tempmask is a mask on the
    photon index whereas mask is the actual aperture mask of our sensor
% photonPhaseTotal=photonPhaseTotal(tempmask,:);
% clear tempmask
nmask=numel(sensorDistance);
for whichmask=1:nmask
    if whichmask==1
        N1=NOuter;
    else
        N1=NInner(whichmask-1);
    end
    N2=NInner(whichmask);
    indx=sub2ind(size(mask{whichmask}),ceil(photonPhaseTotal(:,Y)/maskelemen
    tsize(whichmask,1)),ceil(photonPhaseTotal(:,Z)/maskelements(whichmask
    ,2)));
    transparencymask=mask{whichmask}(indx);
    photonPhaseTotal(:,7)=photonPhaseTotal(:,7).*transparencymask;
    photonPhaseTotal=photonPhaseTotal((photonPhaseTotal(:,7)>1e-5),:); %if
    the mask renders the photon extremely attenuated, just forget about it.
    allows much better speeds for masks with opaque elements
    thetadegOuter=real(acosd(-photonPhaseTotal(:,VX))); %this is the angle
    in degrees with respect to normal, matters for pixel vignetting
    phideg=real(acosd(photonPhaseTotal(:,VY)./max(sind(thetadegOuter),eps)).
    *sign(photonPhaseTotal(:,VZ))); %eps to avoid divide by zero... any
    phideg term gets multiplied by sin(thetadeg) so errors introduced are

```

```

zero'd out
thetadegInner=real(asind(N1/N2*sind(thetadegOuter))); %refract as you
pass through the pinholes, from brain tissue to standoff material.
if whichmask<nmask
    p=(sensordistance(whichmask)-
sensordistance(whichmask+1))./cosd(thetadegInner);
else
    p=sensordistance(whichmask)./cosd(thetadegInner);
end
if whichmask==nmask
    if slope>=0;
        slopeeff=1-(slope)*abs(thetadegInner); %positive slope uses
approximate linear (in angle) vignetting
    else
        warning('negative slope detected, negative slopes not usually
used with vpvmaskgen3')
        slopeeff=(abs(thetadegInner)<abs(slope)); %negative slope flags
this as a cutoff...photons closer to normal than the cutoff angle are
unaffected; photons more oblique are zero'd out
    end
end
photonPhaseTotal(:,VX)=-cosd(thetadegInner);
photonPhaseTotal(:,VY)=sind(thetadegInner).*cosd(phideg);
photonPhaseTotal(:,VZ)=sind(thetadegInner).*sind(phideg);
photonPhaseTotal(:,Y)=photonPhaseTotal(:,Y)+p.*photonPhaseTotal(:,VY);
photonPhaseTotal(:,Z)=photonPhaseTotal(:,Z)+p.*photonPhaseTotal(:,VZ);
donteval=photonPhaseTotal(:,Y)<=0 | photonPhaseTotal(:,Y)>wy |
photonPhaseTotal(:,Z)<=0 | photonPhaseTotal(:,Z)>wz | slopeeff<=0;
photonPhaseTotal=photonPhaseTotal(~donteval,:);
end

ypp=ceil(photonPhaseTotal(:,Y)/py);
zpp=ceil(photonPhaseTotal(:,Z)/pz);
weights=photonPhaseTotal(:,WEIGHT);
slopeeff=slopeeff(~donteval);

for a=1:numel(ypp)
    vPixelVig(ypp(a),zpp(a))=vPixelVig(ypp(a),zpp(a))+weights(a)*slopeeff(a);
end
end

```