pulse2percept: A Python-based simulation framework for bionic vision

Michael Beyeler[‡§¶*], Geoffrey M. Boynton[‡], Ione Fine[‡], Ariel Rokem[¶§]

https://youtu.be/KxsNAa-P2X4

Abstract—By 2020 roughly 200 million people worldwide will suffer from photoreceptor diseases such as retinitis pigmentosa and age-related macular degeneration, and a variety of retinal sight restoration technologies are being developed to target these diseases. One technology, analogous to cochlear implants, uses a grid of electrodes to stimulate remaining retinal cells. Two brands of retinal prostheses are currently approved for implantation in patients with late stage photoreceptor disease. Clinical experience with these implants has made it apparent that the vision restored by these devices differs substantially from normal sight. To better understand the outcomes of this technology, we developed *pulse2percept*, an open-source Python implementation of a computational model that predicts the perceptual experience of retinal prosthesis patients across a wide range of implant configurations. A modular and extensible user interface exposes the different building blocks of the software, making it easy for users to simulate novel implants, stimuli, and retinal models. We hope that this library will contribute substantially to the field of medicine by providing a tool to accelerate the development of visual prostheses.

Index Terms—bionic vision, retinal implant, pulse2percept, prosthesis

Introduction

Two frequent causes of blindness in the developed world are age-related macular degeneration (AMD) and retinitis pigmentosa (RP) [BBB+84], [Gro04]. Both of these diseases have a hereditary component, and are characterized by a progressive degeneration of photoreceptors in the retina that lead to gradual loss of vision.

Microelectronic retinal prostheses have been developed in an effort to restore sight to RP and AMD patients. Analogous to cochlear implants, these devices function by electrically stimulating surviving retinal neurons in order to evoke neuronal responses that are transmitted to the brain and interpreted by patients as visual percepts (Fig. 1). Two of these devices are already approved for commercial use, and a number of other companies have either started or are planning to start clinical trials of devices in the near future. Other types of technologies, such as optogenetics and genetic modification are also areas of active research. Blinded individuals may potentially be offered a wide range of sight restoration options within a decade [FCL15].

* Corresponding author: mbeyeler@uw.edu
‡ Department of Psychology, University of Washington
§ Institute for Neuroengineering, University of Washington
¶ eScience Institute, University of Washington

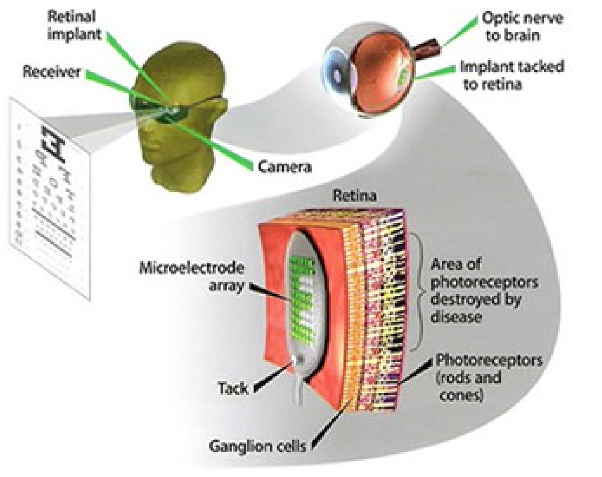


***Fig. 1:** Electronic retinal prosthesis. Light from the visual scene is captured by an external camera and transformed into electrical pulses delivered through microelectrodes to stimulate the retina.*

One major challenge in the development of retinal prostheses is predicting what patients will see when they use their devices. Interactions between implant electronics and the underlying neurophysiology cause nontrivial perceptual distortions in both space and time [FB15], [BRBFss] that severely limit the quality of the generated visual experience.

Our goal was to develop a simulation framework that could describe the visual percepts of retinal prosthesis patients over space and time. We refer to these simulations as 'virtual patients', analogous to the virtual prototyping that has proved so useful in other complex engineering applications.

Here we present an open-source implementation of these models as part of *pulse2percept*, a BSD-licensed Python-based simulation framework [BSD17] that relies on the NumPy and SciPy stacks, as well as contributions from the broader Python community. Based on the detailed specification of a patient's implant configuration, and given a desired electrical stimulus, the model predicts the perceptual distortions experienced by 'virtual patients' over both space and time.

We hope that this library will contribute substantially to the field of medicine by providing a tool to accelerate the development of visual prostheses. Researchers may use this tool to improve stimulation protocols of existing implants or to aid development of future devices. In addition, this tool might guide government agencies, such as the FDA and Medicare, in making reimbursement decisions. Furthermore, this tool can be used to guide patients and doctors in their decision as to when or whether to be implanted, and which device to select.

The remainder of this paper is organized as follows: We start by introducing the neuroscience background necessary to understand the interactions between implant electronics and the underlying neurophysiology. We then detail the computational model that underlies *pulse2percept*, before we give a simple usage example and go into implementation details. We then review our solutions to various technical challenges, and conclude by discussing the broader impact for this work for computational neuroscience and neural engineering communities in more detail.

Background

The first steps in seeing begin in the retina, where a mosaic of photoreceptors converts incoming light into electrochemical signals that encode the intensity of light as a function of position (two dimensions), wavelength, and time [Rod98]. The electrochemical signal is passed on to specialized neuronal circuits consisting of a variety of cell types (such as bipolar, amacrine, and horizontal cells), which act as feature detectors for basic sensory properties, such as spatial contrast and temporal frequency. These sensory features are then encoded in parallel across approximately 1.5 million retinal ganglion cells, which form the output layer of the retina. Each ganglion cell relays the electrical signal to the brain via a long axon fiber that passes from the ganglion cell body to the optic nerve and on to the brain.

Diseases such as RP and AMD are characterized by a progressive degeneration of photoreceptors, gradually affecting other layers of the retina [HPdJ+99], [MNS08]. In severe end-stage RP, roughly 95% of photoreceptors, 20% of bipolar cells, and 70% of ganglion cells degenerate [SHd+97]. In addition, the remaining cells undergo corruptive re-modeling in late stages of the disease [MJWS03], [MJ03], so that little or no useful vision is retained.

Microelectronic retinal prostheses have been developed in an effort to restore sight to individuals suffering from RP or AMD. Analogous to cochlear implants, the goal of retinal prostheses is to electrically stimulate surviving bipolar or ganglion cells in order to evoke neuronal responses that are interpreted by the brain as visual percepts. The electrical stimulus delivers charge to the cell membrane that depolarizes the neuron and opens voltage-sensitive ion channels. This bypasses the natural presynaptic neurotransmitter excitation and causes the activated neurons to stimulate their postsynaptic targets. Therefore, selective spatiotemporal modulation of retinal neurons with an array of electrodes should allow a prosthesis to coordinate retinal activity in place of natural photoreceptor input [WWH16].

Several types of retinal prostheses are currently in development. These vary in their user interface, light-detection method, signal processing, and microelectrode placement within the retina (for a recent review see [WWH16]). As far as our model is concerned, the critical factor is the placement of the microelectrodes on the retina (Fig. 2). The three main locations for microelectrode implant placement are *epiretinal* (i.e., on top of the retinal surface, above the ganglion cells), *subretinal* (i.e., next to the bipolar cells in the space of the missing photoreceptors), and *suprachoroidal* (i.e., between the choroid and the sclera). Each of these approaches is similar in that light from the visual scene is captured via a camera and transformed into electrical pulses delivered through electrodes to stimulate the retina.

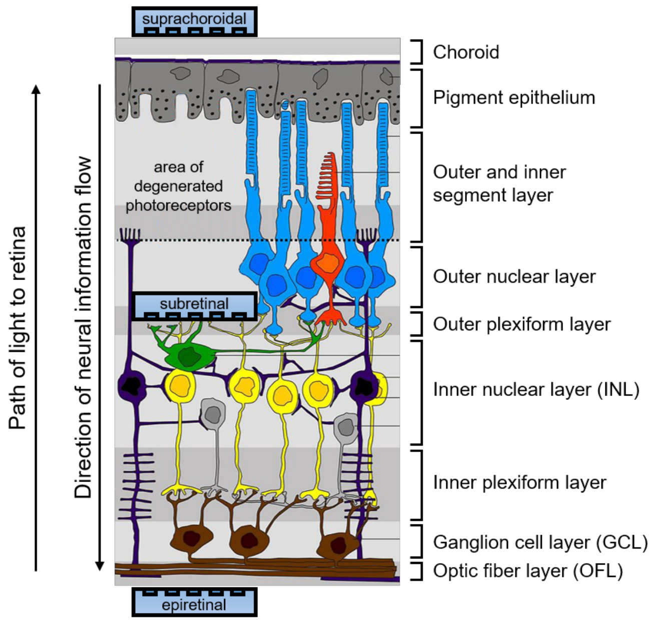


Fig. 2: *Diagram of the retina and common locations of retinal prosthesis microelectrode arrays. Retinitis pigmentosa causes severe photoreceptor degeneration. Epiretinal electrode arrays are placed in the vitreous, next to the optic fiber layer (OFL). Subretinal arrays are placed by creating a space between the choroid and remaining retinal tissue. Suprachoroidal arrays are placed behind the choroid.* pulse2percept *allows for simulation of processing in the inner nuclear layer (INL), ganglion cell layer (GCL), and optic fiber layer (OFL). Based on "Retina layers" by Peter Hartmann, CC BY-SA 3.0.*

As mentioned above, two devices are currently approved for commercial use and are being implanted in patients across the US and Europe: the Argus II device (epiretinal, Second Sight Medical Products Inc., [dCDH+16]) and the Alpha-IMS system (subretinal, Retina Implant AG, [SBSB+15]). At the same time, a number of other companies have either started or are planning to start clinical trials in the near future, potentially offering a wide range of sight restoration options for the blind within a decade [FCL15].

However, clinical experience with existing retinal prostheses makes it apparent that the vision provided by these devices differs substantially from normal sight. Interactions between implant electronics and the underlying neurophysiology cause nontrivial perceptual distortions in both space and time [FB15], [BRBFss] that severely limit the quality of the generated visual experience. For example, stimulating a single electrode rarely produces the experience of a ‘dot’ of light, instead leading to percepts that vary drastically in shape, varying in description from ‘blobs’, to ‘streaks’ and ‘half-moons’. The percept produced by stimulating a single electrode with a continuous pulse train also fades over time, usually disappearing over a course of seconds. As a result, patients do not report seeing an interpretable world. One patient describes it as like: *"... looking at the night sky where you have millions of twinkly lights that almost look like chaos"* [Pre15].

Previous work by our group has focused on development of computational models to describe some of these distortions for a small number of behavioral observations in either space [NFH+12] or time [HGW+09]. Here we present a model that can describe spatial distortions, temporal nonlinearities, and spatiotemporal interactions reported across a wide range of conditions, devices, and patients.

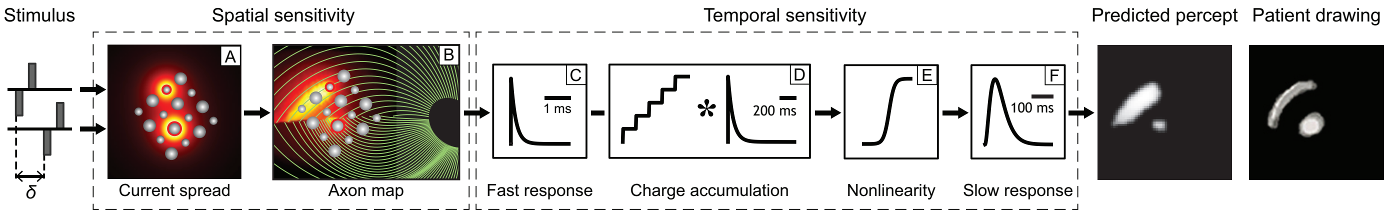


Fig. 3: Full model cascade. A simulated electrical stimulus is processed by a series of linear filtering and nonlinear processing steps that model the spatial (A, B) and temporal sensitivity (C-F) of the retinal tissue. An Argus I device is shown (16 electrodes of 260 or 520 microns diameter arranged in a checkerboard pattern). The output of the model is a prediction of the visual percept in both space and time (example frame shown), which can be compared to human patients' drawings.

Computational Model of Bionic Vision

Analogous to models of cochlear implants [Sha89], the goal of our computational model is to approximate, via a number of linear filtering and nonlinear processing steps, the neural computations that convert electrical pulse trains across multiple electrodes into a perceptual experience in space and time.

Model parameters were chosen to fit data from a variety of behavioral experiments in patients with prosthetic devices. For example, in threshold experiments patients were asked to report whether or not they detected a percept. Across many trials, the minimum stimulation current amplitude needed to reliably detect the presence of a percept on 80% of trials was found. This threshold was measured across a variety of pulse trains that varied across dimensions such as frequency, duty cycle, and duration. In other experiments patients reported the apparent brightness or size of percepts on a rating scale. In others patients drew the shapes of the percepts evoked by stimulation. The model has been shown to generalize across individual electrodes, patients, and devices, as well as across different experiments. Detailed methods of how the model was validated can be found in [HGW$^+$09], [NFH$^+$12], [BRBFss]. Here we provide a brief overview of the model cascade.

The full model cascade for an Argus I epiretinal prosthesis is illustrated in Fig. 3. The Argus I device simulated here consists of electrodes of 260 μm and 520 μm diameter, arranged in a checkerboard pattern (Fig. 3 A). In this example, input to the model is a pair of simulated pulse trains phase-shifted by δ ms, which are delivered to two individual simulated electrodes.

The first stages of the model describe the spatial distortions resulting from interactions between the electronics and the neuroanatomy of the retina. We assume that the current density caused by electrical stimulation decreases as a function of distance from the edge of the electrode [ABK$^+$08]:

$$c(d) = \frac{\alpha}{\alpha + d^n} \tag{1}$$

where d is the 3D Euclidean distance to the electrode edge, $\alpha = 14000$ and the exponent $n = 1.69$. Current fields for two stimulated electrodes are shown, Fig. 3 A (the hotter the color, the higher the current).

Due to the fact that epiretinal implants sit on top of the optic fiber layer (Fig. 2), they do not only stimulate ganglion cell bodies but also ganglion cell axons. It is critical to note that, perceptually, activating an axon fiber that passes under a stimulated electrode is indistinguishable from the percept that would be elicited by activating the corresponding ganglion cell *body*. The produced visual percept will appear in the spatial location in visual space for which the corresponding ganglion cell and axon usually encodes information. Ganglion cells send their axons on highly stereotyped pathways to the optic disc (green lines in Fig. 3 B), which have been mathematically described [JNS$^+$09]. As a result, electrical stimulation of axon fibers leads to predictable visual 'streaks' or 'comet trails' that are aligned with the axonal pathways.

We therefore convert the spatial map of current densities into a tissue activation map by accounting for axonal stimulation. We model the sensitivity of axon fibers as decreasing exponentially as a function of distance from the corresponding ganglion cell bodies. The resulting tissue activation map across the retinal surface is shown as a heatmap in Fig. 3 B (the hotter the color, the larger the amount of tissue stimulation).

The remaining stages of the model describe temporal nonlinearities. Every pixel of the tissue activation map is modulated over time by the applied electrical pulse train in order to predict a perceived brightness value that varies over time. This involves applying a series of linear filtering (Fig. 3 C, D, and F) and nonlinear processing (Fig. 3 E) steps in the time domain that are designed to approximate the processing of visual information within the retina and visual cortex.

Linear responses in Fig. 3 C, D, and F are modeled as temporal low-pass filters, or 'leaky integrators', using gamma functions of order n:

$$\delta(t,n,\tau) = \frac{\exp(-t/\tau)}{\tau(n-1)!}\left(\frac{t}{\tau}\right)^{n-1} \tag{2}$$

where t is time, n is the number of identical, cascading stages, and τ is the time constant of the filter.

The first temporal processing step convolves the timeseries of tissue activation strengths $f(t)$ at a particular spatial location with a one-stage gamma function ($n = 1$, time constant $\tau_1 = 0.42$ ms) to model the impulse response function of retinal ganglion cells (Fig. 3 C):

$$r_1(t) = f(t) * \delta(t,1,\tau_1), \tag{3}$$

where $*$ denotes convolution.

Behavioral data suggests that the system becomes less sensitive as a function of accumulated charge. This is implemented by calculating the amount of accumulating charge at each point of time in the stimulus, $c(t)$, and by convolving this accumulation with a second one-stage gamma function ($n = 1$, time constant $\tau_2 = 45.3$ ms; Fig. 3 D). The output of this convolution is scaled by a factor $\varepsilon_1 = 8.3$ and subtracted from r_1 (Eq. 3):

$$r_2(t) = r_1(t) - \varepsilon_1\big(c(t) * \delta(t,1,\tau_2)\big). \tag{4}$$

The response $r_2(t)$ is then passed through a stationary nonlinearity (Fig. 3 E) to model the nonlinear input-output relationship of ganglion cell spike generation:

$$r_3(t) = r_2(t)\frac{\alpha}{1+\exp\frac{\delta - \max_t r_2(t)}{s}} \tag{5}$$

where $\alpha = 14$ (asymptote), $s = 3$ (slope), and $\delta = 16$ (shift) are chosen to match the observed psychophysical data.

Finally, the response $r_3(t)$ is convolved with another low-pass filter described as a three-stage gamma function ($n = 3$, $\tau_3 = 26.3$ ms) intended to model slower perceptual processes in the brain (Fig. 3 F):

$$r_4(t) = \varepsilon_2 r_3(t) * \delta(t, 3, \tau_3), \tag{6}$$

where $\varepsilon_2 = 1000$ is a scaling factor used to scale the output to subjective brightness values in the range [0, 100].

The output of the model is thus a movie of brightness values that corresponds to the predicted perceptual experience of the patient. An example percept generated is shown on the right-hand side of Fig. 3 ('predicted percept', brightest frame in the movie).

Implementation and Results

Code Organization

The *pulse2percept* project seeks a trade-off between optimizing for computational performance and ease of use. To facilitate ease of use, we organized the software as a standard Python package, consisting of the following primary modules:

- `api`: a top-level Application Programming Interface.
- `implants`: implementations of the details of different retinal prosthetic implants. This includes Second Sight's Argus I and Argus II implants, but can easily be extended to feature custom implants (see Section Extensibility).
- `retina`: implementation of a model of the retinal distribution of nerve fibers, based on [JNS+09], and an implementation of the temporal cascade of events described in Eqs. 2-6. Again, this can easily be extended to custom temporal models (see Section Extensibility).
- `stimuli`: implementations of commonly used electrical stimulation protocols, including methods for translating images and movies into simulated electrical pulse trains. Again, this can easily be extended to custom stimulation protocols (see Section Extensibility).
- `files`: a means to load/store data as images and videos.
- `utils`: various utility and helper functions.

Basic Usage

Here we give a minimal usage example to produce the percept shown on the right-hand side of Fig. 3.

Convention is to import the main `pulse2percept` module as `p2p`. Throughout this paper, if a class is referred to with the prefix `p2p`, it means this class belongs to the main pulse2percept library (e.g., `p2p.retina`):

```python
import pulse2percept as p2p
```

`p2p.implants`: Our goal was to create electrode implant objects that could be configured in a highly flexible manner. As far as placement is concerned, an implant can be placed at a particular location on the retina (`x_center`, `y_center`) with respect to the fovea (in microns), and rotated as you see fit (`rot`):

```python
import numpy as np
implant = p2p.implants.ArgusI(x_center=-800,
                              y_center=0,
                              h=80,
                              rot=np.deg2rad(35))
```

Here, we make use of the `ArgusI` array type, which provides predefined values for array type ('epiretinal') and electrode diameters. In addition, the distance between the array and the retinal tissue can be specified via the height parameter (`h`), either on a per-electrode basis (as a list) or using the same value for all electrodes (as a scalar).

The electrodes within the implant can also be specified. An implant is a wrapper around a list of `p2p.implants.Electrode` objects, which are accessible via indexing or iteration (e.g., via `[for i in implant]`). This allows for specification of electrode diameter, position, and distance to the retinal tissue on a per-electrode basis. Once configured, every `Electrode` object in the implant can also be assigned a name (in the Argus I implant, they are A1 - A16; corresponding to the names that are commonly used by Second Sight Medical Products Inc.). The first electrode in the implant can be accessed both via its index (`implant[0]`) and its name (`implant['A1']`).

Once the implant is created, it can be passed to the simulation framework. This is also where you specify the back end (currently supported are 'serial', 'joblib' [Job16], and 'dask' [Das16]):

```python
sim = p2p.Simulation(implant, engine='joblib',
                     num_jobs=8)
```

The simulation framework provides a number of setter functions for the different layers of the retina. These allow for flexible specification of optional settings, while abstracting the underlying functionality.

`p2p.retina`: This includes the implementation of a model of the retinal distribution of nerve fibers, based on [JNS+09], and implementations of the temporal cascade of events described in Eqs. 2-6.

Things that can be set include the spatial sampling rate of the optic fiber layer (`ssample`) as well as the spatial extent of the retinal patch to be simulated (given by the corner points `[xlo, ylo]` and `[xhi, yhi]`). If the coordinates of the latter are not given, a patch large enough to fit the specified electrode array will be automatically selected:

```python
ssample = 100  # microns
num_axon_samples = 801
sim.set_optic_fiber_layer(ssample=ssample,
                          num_axon_samples=801)
```

Similarly, for the ganglion cell layer we can choose one of the pre-existing cascade models and specify a temporal sampling rate:

```python
tsample = 0.005 / 1000  # seconds
sim.set_ganglion_cell_layer('Nanduri2012',
                            tsample=tsample)
```

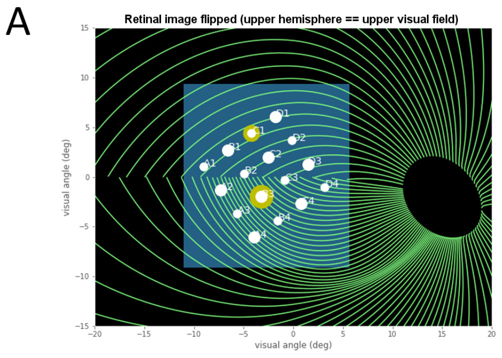
As its name suggest, `'Nanduri2012'` implements the model detailed in [NFH+12]. Other options include `'Horsager2009'` [HGW+09] and `'latest'`.

It's also possible to specify your own (custom) model, see Section Extensibility below.

`p2p.stimuli`: A stimulation protocol can be specified by assigning stimuli from the `p2p.stimuli` module to specific electrodes. An example is to set up a pulse train of particular stimulation frequency, current amplitude and duration. Because of safety considerations, all real-world stimuli must be balanced biphasic pulse trains (i.e., they must have a positive and negative phase of equal area, so that the net current delivered to the tissue sums to zero).

It is possible to specify a pulse train for each electrode in the implant as follows:

```python
# Stimulate two electrodes, others set to zero
stim = []
for elec in implant:
```



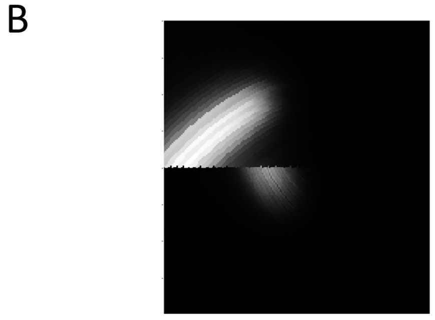


Fig. 4: *Model input/output generated by the example code. (A) An epiretinal Argus I array is placed near the fovea, and two electrodes ('C1' and 'B3') are stimulated with 50 Hz, 20 uA pulse trains. The plot is created by lines 34-36. Note that the retinal image is flipped, so that the upper hemisphere corresponds to the upper visual field. (B) Predicted visual percept (example frame shown). The plot is created by line 41.*

```
19      if elec.name == 'C1' or elec.name == 'B3':
20          # 50 Hz, 20 uA, 0.5 sec duration
21          pt = p2p.stimuli.PulseTrain(tsample,
22                                      freq=50.0,
23                                      amp=20.0,
24                                      dur=0.5)
25      else:
26          pt = p2p.stimuli.PulseTrain(tsample, freq=0)
27      stim.append(pt)
```

However, since implants are likely to have electrodes numbering in the hundreds or thousands, this method will rapidly become cumbersome when assigning pulse trains across multiple electrodes. Therefore, an alternative is to assign pulse trains to electrodes via a dictionary:

```
28 stim = {
29     'C1': p2p.stimuli.PulseTrain(tsample, freq=50.0,
30                                  amp=20.0, dur=0.5)
31     'B3': p2p.stimuli.PulseTrain(tsample, freq=50.0,
32                                  amp=20.0, dur=0.5)
33 }
```

At this point, we can highlight the stimulated electrodes in the array:

```
34 import matplotlib.pyplot as plt
35 %matplotlib inline
36 sim.plot_fundus(stim)
```

The output can be seen in Fig. 4 A.

Finally, the created stimulus serves as input to `sim.pulse2percept`, which is used to convert the pulse trains into a percept. This allows users to simulate the predicted percepts for simple input stimuli, such as stimulating a pair of electrodes, or more complex stimuli, such as stimulating a grid of electrodes in the shape of the letter A.

At this stage in the model it is possible to consider which retinal layers are included in the temporal model, by selecting from the following list (see Fig. 2 for a schematic of the anatomy):

- `'OFL'`: optic fiber layer (OFL), where ganglion cell axons reside,
- `'GCL'`: ganglion cell layer (GCL), where ganglion cell bodies reside, and
- `'INL'`: inner nuclear layer (INL), where bipolar cells reside.

A list of retinal layers to be included in the simulation is then passed to the `pulse2percept` method:

```
37 # From pulse train to percept
38 percept = sim.pulse2percept(stim, tol=0.25,
39                             layers=['GCL', 'OFL'])
```

This allows the user to run simulations that include only the layers relevant to a particular simulation. For example, axonal stimulation and the resulting axon streaks can be ignored by omitting `'OFL'` from the list. By default, all three supported layers are included.

Here, the output `percept` is a `p2p.utils.TimeSeries` object that contains the time series data in its `data` container. This time series consists of brightness values (arbitrary units in [0, 100]) for every pixel in the percept image.

Alternatively, it is possible to retrieve the brightest (mean over all pixels) frame of the time series:

```
40 frame = p2p.get_brightest_frame(percept)
```

Then we can plot it with the help of Matplotlib (Fig. 4 B):

```
41 plt.imshow(frame, cmap='gray')
```

`p2p.files`: *pulse2percept* offers a collection of functions to convert the `p2p.utils.TimeSeries` output into a movie file via scikit-video [Sci17] and ffmpeg [FFm10].

For example, a percept can be stored to an MP4 file as follows:

```
42 # Save movie at 15 frames per second
43 p2p.files.save_video(percept, 'mypercept.mp4',
44                      fps=15)
```

For convenience, *pulse2percept* provides a function to load a video file and convert it to the `p2p.utils.TimeSeries` format:

```
45 # Load video as TimeSeries of size (M x N x T),
46 # M: height, N: width, T: number of frames
47 video = p2p.files.load_video('mypercept.mp4')
```

Analogously, *pulse2percept* also provides functionality to go directly from images or videos to electrical stimulation on an electrode array:

```
48 from_img = p2p.stimuli.image2pulsetrain('myimg.jpg',
49                                         implant)
50 from_vid = p2p.stimuli.video2pulsetrain('mymov.avi',
51                                         implant)
```

These functions are based on functionality provided by scikit-image [S v14] and scikit-video [Sci17], respectively, and come with a number of options to specify whether brightness should be encoded as pulse train amplitude or frequency, at what frame rate to sample the movie, whether to maximize or invert the contrast, and so on.

Extensibility

As described above, our software is designed to allow for implants, retinal models, and pulse trains to be customized. We provide extensibility mainly through mechanisms of class inheritance.

Retinal Implants: Creating a new implant involves inheriting from `p2p.implants.ElectrodeArray` and providing a property `etype`, which is the electrode type (e.g., `'epiretinal'`, `'subretinal'`):

```
52 import pulse2percept as p2p
53
54 class MyImplant(p2p.implants.ElectrodeArray):
55
56     def __init__(self, etype):
57         """Custom electrode array
58
59         Parameters
60         ----------
61         etype : str
62             Electrode type, {'epiretinal',
63             'subretinal'}
64         """
65         self.etype = etype
66         self.num_electrodes = 0
67         self.electrodes = []
```

Then new electrodes can be added by utilizing the `add_electrodes` method of the base class:

```
68 myimplant = MyImplant('epiretinal')
69 r = 150  # electrode radius (um)
70 x, y = 10, 20  # distance from fovea (um)
71 h = 50  # height from retinal surface (um)
72 myimplant.add_electrodes(r, x, y, h)
```

Retinal cell models: Any new ganglion cell model is described as a series of temporal operations that are carried out on a single pixel of the image. It must provide a property called `tsample`, which is the temporal sampling rate, and a method called `model_cascade`, which translates a single-pixel pulse train into a single-pixel percept over time:

```
73 class MyGanglionCellModel(p2p.retina.TemporalModel):
74     def model_cascade(self, in_arr, pt_list, layers,
75                       use_jit):
76         """Custom ganglion cell model
77
78         Parameters
79         ----------
80         in_arr : array_like
81             2D array <#layers x #time points> of
82             effective current values at a single
83             pixel location.
84         pt_list : list
85             List of pulse train `data` containers.
86         layers : list
87             List of retinal layers to simulate.
88             Choose from:
89             - 'OFL': optic fiber layer
90             - 'GCL': ganglion cell layer
91             - 'INL': inner nuclear layer
92         use_jit : bool
93             If True, applies just-in-time (JIT)
94             compilation to expensive computations
95             for additional speedup (requires Numba)
96         """
97         return p2p.utils.TimeSeries(self.tsample,
98                                     in_arr[0, :])
```

This method can then be passed to the simulation framework:

```
99 mymodel = MyGanglionCellModel()
100 sim.set_ganglion_cell_layer(mymodel)
```

It will then automatically be selected as the implemented ganglion cell model when `sim.pulse2percept` is called.

Stimuli: The smallest stimulus building block provided by *pulse2percept* consists of a single pulse of either positive current (anodic) or negative current (cathodic), which can be created via `p2p.stimuli.MonophasicPulse`. However, as described above, any real-world stimulus must consist of biphasic pulses with zero net current. A single biphasic pulse can be created via `p2p.stimuli.BiphasicPulse`. A train of such pulses can be created via `p2p.stimuli.PulseTrain`. This setup gives the user the opportunity to build their own stimuli by creating pulse trains of varying amplitude, frequency, and inter-pulse intervals.

In order to define new pulse shapes and custom stimuli, the user can either inherit from any of these stimuli classes or directly from `p2p.utils.TimeSeries`. For example, a biphasic pulse can be built from two monophasic pulses as follows:

```
101 from pulse2percept.stimuli import MonophasicPulse
102
103 class MyBiphasicPulse(p2p.utils.TimeSeries):
104
105     def __init__(self, pdur, tsample):
106         """A charge-balanced pulse with a cathodic
107            and anodic phase
108
109         Parameters
110         ----------
111         tsample : float
112             Sampling time step in seconds.
113         pdur : float
114             Single-pulse phase duration (seconds).
115         """
116         on = MonophasicPulse('anodic', pdur, tsample,
117                              0, pdur)
118         off = MonophasicPulse('cathodic', pdur,
119                               tsample, 0, pdur)
120         on.append(off)
121         utils.TimeSeries.__init__(self, tsample, on)
```

Implementation Details

pulse2percept's main technical challenge is computational cost: the simulations require a fine subsampling of space, and span several orders of magnitude in time. In the space domain the software needs to be able to simulate electrical activation of individual retinal ganglion cells on the order of microns. In the time domain the model needs to be capable of dealing with pulse trains containing individual pulses on the sub-millisecond time scale that last over several seconds.

Like the brain, we solve this problem through parallelization. Spatial interactions are confined to an initial stage of processing (Fig. 3 A, B), after which all spatial computations are parallelized using two back ends (Joblib [Job16] and Dask [Das16]), with both multithreading and multiprocessing options.

However, even after parallelization, computing the temporal response remains a computational bottleneck. Initial stages of the temporal model require convolutions of arrays (e.g., Eqs. 2 and 3) that describe responses of the model at high temporal resolution (sampling rates on the order of 10 microseconds) for pulse trains lasting for at least 500 milliseconds. These numerically-heavy sections of the code are sped up using a conjunction of three strategies. First, as described above, any given electrode generally only stimulates a subregion of the retina. As a consequence, when only a few electrodes are active, we can often obtain substantial speed savings by ignoring pixels which are not significantly stimulated by any electrode (see tolerance parameter `tol` on line 38 of the example code). Second, electrical stimulation is often carried out at relatively low pulse train frequencies of less than 50 Hz. Since the individual pulses within the pulse train are usually very short (~75-450 microseconds), input pulse trains tend to be sparse. We can exploit this fact to speed up computation time by avoiding direct convolution with the entire time-series whenever possible,

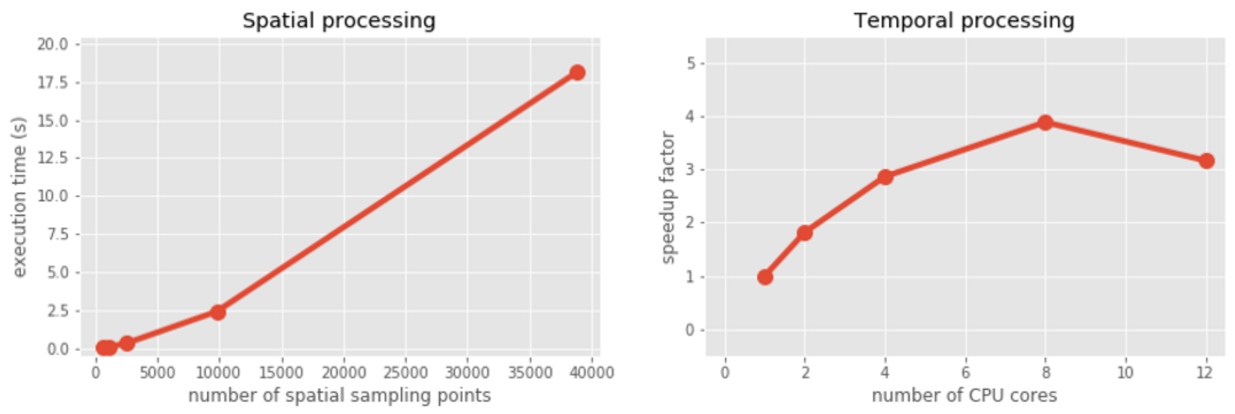


Fig. 5: *Computational performance. (A) Compute time to generate an 'effective stimulation map' is shown as a function of the number of spatial sampling points used to characterize the retina. (B) Speedup factor (serial execution time / parallel execution time) is shown as a function of the number of CPU cores. Execution times were collected for the an Argus II array (60 electrodes) simulating the letter 'A' (roughly 40 active electrodes, 20 Hz/20 uA pulse trains) over a period of 500 ms (*`tsample` *was 10 microseconds,* `ssample` *was 50 microns). Joblib and Dask parallelization back ends gave similar results.*

and instead relying on a custom-built sparse convolution function. Preprocessing of sparse pulse train arrays allows us to carry out temporal convolution only for those parts of the time-series that include nonzero current amplitudes. Finally, these convolutions are sped up wih LLVM-base compilation implemented using Numba [LPS15].

Computational Performance

We measured computational performance of the model for both spatial and temporal processing using a 12-core Intel Core i7-5930K operating at 3.50 GHz (64GB of RAM).

The initial stages of the model scale as a function of the number of spatial sampling points in the retina, as shown in Fig. 5 A. Since the spatial arrangement of axonal pathways does not depend on the stimulation protocol or retinal implant, *pulse2percept* automatically stores and re-uses the generated spatial map depending on the specified set of spatial parameters.

The remainder of the model is carried out in parallel, with the resulting speedup factor shown in Fig. 5 B. Here, the speedup factor is calculated as the ratio of single-core execution time and parallel execution time. On this particular machine, the maximum speedup factor is obtained with eight cores, above which the simulation shifts from being CPU bound to being memory bound, leading to a decrease in speedup. At its best, simulating typical stimulation of an Argus II over a timecourse of 500 milliseconds (at 50 microns spatial resolution and 10 ms temporal resolution) required 79 seconds of execution time. According to line profiling, most of the time is spent executing the slow convolutions (Fig. 3 D, F), thus heavily relying on the computational performance of the SciPy implementation of the Fast Fourier Transform. Due to the current design constraints (see Discussion), the software is better suited for rapid prototyping rather than real-time execution - although we aim to alleviate this in the future through GPU parallelization (via CUDA [KPL$^+$12] and Dask [Das16]) and cluster computing (via Spark [Apa16]).

Software availability and development

All code can be found at https://github.com/uwescience/pulse2percept, with up-to-date source code documentation available at https://uwescience.github.io/pulse2percept. In addition, the latest stable release is available on the Python Package Index and can be installed using pip:

```
$ pip install pulse2percept
```

The library's test suite can be run as follows:

```
$ py.test --pyargs pulse2percept --doctest-modules
```

All code presented in this paper is current as of the v0.2 release.

Discussion

We presented here an open-source, Python-based framework for modeling the visual processing in retinal prosthesis patients. This software generates a simulation of the perceptual experience of individual prosthetic users - a 'virtual patient'.

The goal of *pulse2percept* is to provide open-source simulations that can allow any user to evaluate the perceptual experiences likely to be produced across both current and future devices. Specifically, the software is designed to meet four software design specifications:

- *Ease of use*: The intended users of this simulation include researchers and government officials who collect or assess perceptual data on prosthetic implants. These researchers are likely to be MDs rather than computer scientists, and might therefore lack technical backgrounds in computing. In the future, we plan for *pulse2percept* to become the back end of a web application similar to [KDM$^+$ss].
- *Modularity*: As research continues in this field, it is likely that the underlying computational models converting electrical stimulation to patient percepts will improve. The modular design of the current release makes it easy to update individual components of the model.
- *Flexibility*: *pulse2percept* allows for rapid prototyping and integration with other analysis or visualization libraries from the Python community. It allows users to play with parameters, and use the ones that fit their desired device. Indeed, within most companies the specifications of implants currently in design is closely guarded intellectual property.
- *Extensibility*: The software can easily be extended to include custom implants, stimulation protocols, and retinal models.

As a result of these design considerations, *pulse2percept* has a number of potential uses.

Device developers can use virtual patients to get an idea of how their implant will perform even before a physical prototype has been created. This is reminiscent of the practice of virtual prototyping in other engineering fields. It becomes possible to make predictions about the perceptual consequences of individual design considerations, such as specific electrode geometries and stimulation protocols. As a result, virtual patients provide a useful tool for implant development, making it possible to rapidly predict vision across different implant configurations. We are currently collaborating with two leading manufacturers to validate the use of this software for both of these purposes.

Virtual patients can also play an important role in the wider community. To a naive viewer, simulations of prosthetic vision currently provided by manufacturers and the press might provide misleading visual outcomes, because these simulations do not take account of the substantial distortions in space and time that are observed by actual patients. On our website we provide example stimulations of real-world vision based on the *pulse2percept* virtual patient.

Prosthetic implants are expensive technology - costing roughly $100k per patient. Currently, these implants are reimbursed on a trial-by-trial basis across many countries in Europe, and are only reimbursed in a subset of states in the USA. Hence our simulations can help guide government agencies such as the FDA and Medicare in reimbursement decisions. Most importantly, these simulations can help patients, their families, and doctors make an informed choice when deciding at what stage of vision loss a prosthetic device would be helpful.

Acknowledgments

Supported by the Washington Research Foundation Funds for Innovation in Neuroengineering and Data-Intensive Discovery (M.B.), by a grant from the Gordon & Betty Moore Foundation and the Alfred P. Sloan Foundation to the University of Washington eScience Institute Data Science Environment (M.B. and A.R.), and by the National Institutes of Health (NEI EY-012925 to G.M.B., EY-014645 to I.F.). Research credits for cloud computing were provided by Amazon Web Services.

References

[ABK+08] AK Ahuja, MR Behrend, M Kuroda, MS Humayun, and JD Weiland. An *in vitro* model of a retinal prosthesis. *IEEE Transactions on Biomedical Engineering*, 55(6):1744–1753, 2008.

[Apa16] Apache Spark Development Team. *Apache Spark: a fast and general cluster computing system for Big Data*, 2016. URL: https://spark.apache.org/releases/spark-release-2-0-0.html.

[BBB+84] CH Bunker, EL Berson, WC Bromley, RP Hayes, and TH Roderick. Prevalence of retinitis pigmentosa in maine. *American Journal of Ophthalmology*, 97:357–365, 1984.

[BRBFss] M Beyeler, A Rokem, GM Boynton, and I Fine. Learning to see again: Biological constraints on cortical plasticity and the implications for sight restoration technologies. *Journal of Neural Engineering*, in press. doi:10.1088/1741-2552/aa795e.

[BSD17] Berkeley Software Distribution (BSD) 3-clause "New" or "Revised" License, version 3, 2017. URL: https://github.com/uwescience/pulse2percept/blob/master/LICENSE.

[Das16] Dask Development Team. *Dask: Library for dynamic task scheduling*, 2016. URL: http://dask.pydata.org.

[dCDH+16] L da Cruz, JD Dorn, MS Humayun, G Dagnelie, J Handa, P-O Barale, J-A Sahel, PE Stanga, F Hafezi, and AB Safran et al. Five-year safety and performance results from the argus {II} retinal prosthesis system clinical trial. *Ophthalmology*, 123(10):2248 – 2254, 2016.

[FB15] I Fine and GM Boynton. Pulse trains to percepts: the challenge of creating a perceptually intelligible world with sight recovery technologies. 370(1677), 2015.

[FCL15] I Fine, C Cepko, and MS Landy. Vision Research special issue: Sight restoration: Prosthetics, optogenetics and gene therapy. *Vision Research*, 111:115–123, 2015.

[FFm10] FFmpeg Development Team. *FFmpeg: A complete, cross-platform solution to record, convert and stream audio and video*, 2010. URL: http://www.ffmpeg.org.

[Gro04] The Eye Diseases Prevalence Research Group*. Prevalence of age-related macular degeneration in the United States. *Archives of Ophthalmology*, 122(4):564–572, 2004.

[HGW+09] A Horsager, SH Greenwald, JD Weiland, MS Humayun, RJ Greenberg, MJ McMahon, GM Boynton, and I Fine. Predicting visual sensitivity in retinal prosthesis patients. *Investigative Ophthalmology & Visual Science*, 50(4):1483, 2009.

[HPdJ+99] M S Humayun, M Prince, E de Juan, Jr, Y Barron, M Moskowitz, I B Klock, and A H Milam. Morphometric analysis of the extramacular retina from postmortem eyes with retinitis pigmentosa. *Investigative Ophthalmology & Visual Science*, 40(1):143, 1999.

[JNS+09] NM Jansonius, J Nevalainen, B Selig, LM Zangwill, PA Sample, WM Budde, JB Jonas, WA Lagrèze, PJ Airaksinen, and R Vonthein et al. A mathematical description of nerve fiber bundle trajectories and their variability in the human retina. *Vision research*, 49(17):2157–2163, 2009.

[Job16] Joblib Development Team. *Joblib*, 2016. URL: http://pythonhosted.org/joblib/.

[KDM+ss] A Keshavan, E Datta, IM McDonough, CR Madan, K Jordan, and RG Henry. Mindcontrol: A web application for brain segmentation quality control. *NeuroImage*, in press. doi:10.1016/j.neuroimage.2017.03.055.

[KPL+12] A Klöckner, N Pinto, Y Lee, B Catanzaro, P Ivanov, and A Fasih. PyCUDA and PyOpenCL: A Scripting-Based Approach to GPU Run-Time Code Generation. *Parallel Computing*, 38(3):157–174, 2012.

[LPS15] SK Lam, A Pitrou, and S Seibert. Numba: A llvm-based python jit compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, LLVM '15, pages 7:1–7:6, New York, NY, USA, 2015. ACM.

[MJ03] RE Marc and BW Jones. Retinal remodeling in inherited photoreceptor degenerations. *Molecular Neurobiology*, 28:139–147, 2003.

[MJWS03] RE Marc, BW Jones, CB Watt, and E Strettoi. Neural remodeling in retinal degeneration. *Progress in Retinal and Eye Research*, 22:607–655, 2003.

[MNS08] F Mazzoni, E Novelli, and E Strettoi. Retinal ganglion cells survive and maintain normal dendritic morphology in a mouse model of inherited photoreceptor degeneration. *Journal of Neuroscience*, 28:14282–14292, 2008.

[NFH+12] D Nanduri, I Fine, A Horsager, GM Boynton, MS Humayun, RJ Greenberg, and JD Weiland. Frequency and amplitude modulation have different effects on the percepts elicited by retinal stimulation. *Investigative Ophthalmology & Visual Science*, 53(1):205–214, 2012.

[Pre15] Pioneer Press. 'bionic eye' helps duluth man make out his dark world. *Twin Cities Pioneer Press*, 2015.

[Rod98] RW Rodieck. *The first steps in seeing*. Sinauer Associates, 1998.

[S v14] S van der Walt, JL Schönberger, J Nunze-Iglesias, F Boulogne, JD Warner, N Yager, E Gouillart, T Yu and scikit-image contributors. scikit-image: Image processing in Python. *PeerJ*, 2(e453), 2014.

[SBSB+15] K Stingl, KU Bartz-Schmidt, D Besch, CK Chee, CL Cottriall, F Gekeler, M Groppe, TL Jackson, RE MacLaren, and A Koitschev et al. Subretinal visual implant alpha {IMS} – clinical trial interim report. *Vision Research*, 111, Part B:149 – 160, 2015.

[Sci17] Scikit-Video Development Team. *Scikit-Video: Video processing in Python*, 2017. URL: http://www.scikit-video.org.

[Sha89] RV Shannon. A model of threshold for pulsatile electrical stimulation. *Hearing Research*, 40:197–2014, 1989.

[SHd+97] A Santos, MS Humayun, E de Juan Jr., RJ Greenburg, MJ Marsh, IB Klock, and AH Milam. Preservation of the inner retina in retinitis pigmentosa. *Archives of Ophthalmology*, 115:511–515, 1997.

[WWH16] JD Weiland, ST Walston, and MS Humayun. Electrical stimulation of the retina to produce artificial vision. *Annual Review of Vision Science*, 2:273–294, 2016.