Virtual Retina - tutorial

Adrien Wohrer

May 14, 2008

This tutorial provides an introduction to the Virtual Retina software. The goal of this document is not to provide explanations for the underlying retinal model; these can be found in the related article [1], available at https://hal.inria.fr/inria-00160716/en/. It rather describes the architecture of the package, the command lines to be used, and how to configure an xml definition file for the program.

Contents

1 General architecture of the package

1.1 Goals, scope of use

These programs were written to simulate retinal processing on an input video sequence, in a -hopefully- customizable fashion ranking from simple spatio-temporal linear filtering on the input sequence to a precise 'biologically-inspired' model that can include:

1. emission of spike trains by arrays of up to (at least) 100,000 ganglion cells. But... you might as well ask for a single, spiking ganglion cell to reproduce physiological recordings. Or for no spiking cell at all, but rather analogic retinal outputs.
2. modeling of parvocellular and/or magnocellular ganglion cells.
3. a dynamical model of contrast gain control, carried by a shunting inhibition from conductances in the membranes of bipolar cells. The model can also be applied as a luminance gain control between light receptors and horizontal cells.
4. a radial non-homogeneity of the whole retina, with a uniform fovea and a periphery with increasing blur and inversely decreasing ganglion cell density.
5. microsaccadic movements on the input sequence.

1.2 Requirements

Our programs make use of other C++ libraries, whose sources are also present in the repository.

- **CImg**, an image processing toolbox.
- **MvaSpike**, an event-driven spike simulator.
- **libxml++**, a C++ wrapper for *libxml2*, the GNU C interface to deal with xml files.
- **xmlParameters++**, a helper library I wrote to load/save parameters from an xml file in a flexible way.

Furthermore, *xmlParameters++* and *VirtualRetina* are compiled and installed thanks to **CMake**, a multi-platform Make tool.

Finally, a compilation option allows to save the emitted spikes in a special binary format called hdf5. The *hdf5* library, due to its size, is not included in the package. Only the wrapper library **mvaspike_hdf5** is included, that allows *MvaSpike* tools to save output spikes in hdf5 format.

1.3 Installation

Automatic installation

File `Retina_Package` is the root of the package. It contains a shell code `download_build_all.sh` which proceeds to an automatic, LOCAL, installation of Virtual Retina and all its required external libraries, for Linux. Typing

```
sh download_build_all.sh
```

succesively performs the following operations:

1. Download software *CMake* (required for installation of Virtual Retina) and install it locally in

   `Retina_Package/External_Libraries/CMake`

2. Download, compile and install locally the external libraries *MvaSpike*, *libxml++* and *xmlParameters++*, in

   `Retina_Package/VirtualRetina/local/lib/`

3. Compile Virtual Retina, install its libraries in the same `lib` directory, and the executables in

   `Retina_Package/VirtualRetina/local/bin/`

Manual installation

You may wish to perform a ROOT installation of the libraries on your machine, or only a partial installation of some of the external libraries. A look at the content of script `download_build_all.sh` might be helpful. In particular, the script contains web addresses of the libraries, and the version numbers used by Virtual Retina. In short:

1. Library *CImg* consists of a single header `CImg.h`, and requires no precompiled library.

2. Libraries *MvaSpike* and *libxml++* are compiled and installed thanks to Autoconf/Automake, with the classical

   ```
   configure
   make
   (make install)
   ```

3. The Makefiles for libraries *xmlParameters++* and *VirtualRetina* are configured thanks to program `cmake`, with compilation options defined in a set of `CMakeLists.txt` files. Compilation options (e.g. paths for external libraries) can also be changed through a visual interface with program `ccmake`.

Using hdf5

If you wish to save the output retinal spikes under the binary hdf5 format:

1. You must have installed the hdf5 library on your machine.

2. Compile the `mvaspike_hdf5/` library, included in the retina package. You must modify the first line of the Makefile (variable `HDF5`) so that `$(HDF5)/bin/h5cc` be the correct path to the binary executable `h5cc`. Then type `make`.

3. In directory `VirtualRetina/`, before launching the `cmake/make` process, type
   `ccmake CMakeLists.txt`
   and set the CMake variable `USE_HDF5` to `ON`.

After compilation, an new option `-hdf5` will be present in executable `Retina` (and derived executables), allowing to save the spike trains to hdf5 format when desired.

Portability

The software has been tested only under Linux, with g++ compiler 3.4 or more. It has not been tested for VisualC++, so we cannot guarantee that it can compile under Windows. At least, we know that the external libraries required by the software are portable, and the CMake commands for compilation of Virtual Retina are also portable.

1.4 Now what's in there?

The root directory for Virtual Retina is `Retina_Package/VirtualRetina/`. Here are its sub-directories:

- `local/bin` contains the compiled executables.
- `src/` contains code sources for the `virtual_retina` library, a library of general tools, and for the executables.
- `test/sequences/` contains (two) test sequences. (wow!)
- `test/retina_files/` contains three sample retina definition files written in xml.
- `tmp/` is the default output directory of all executables.
- `experiments/` contains some experiments on single cells, to validate the underlying retina model.

2 The executables

All executables have a `-h` command that provides the list of options for the executable. In this Section we only present the most important things to know to run the programs.

2.1 The `Retina` executable

This is the main simulation executable. Input is a 3-dimensional sequence (space and time). Output is sets of spike trains or, if you did not ask for spikes, analogic spatio-temporal sequences corresponding to the 'activities' of ganglion cells. It is typically called by:

```
Retina  path/to/my/test_sequence*.pgm  -ret path/to/my/retina.xml
        -r 10  -outD path/to/my/saving_directory
```

with possible options:

- input sequences can be passed as a series of 2d frames under any usual format (as here), or directly as a .inr 3d file.
- `-ret` gives the path to the retina definition file in xml format. This is the file containing the definition of the whole retinal architecture and parameters. Some customized files are already provided in the package, but for a deeper understanding we refer to Section 3 of this tutorial. Amongst other parameters, it fixes the retinal time step (say, here, 5 ms).
- `-r 10` asks that each input frame be presented to the retina for a duration of 10*(retinal time step) = 50 ms in this case.
- `-outD` gives the path to the directory where all simulation files will be saved. By default, this directory is `VirtualRetina/tmp/`.
- Full list of options by typing `-h` when calling the executable. They include: position of the retina in the input sequence, saving options, display options, etc.

The executable has several output files:

simulation.txt is the main output file of the simulation, which can be used as input to executables who need to load an already existing simulation. It contains general information such as: input sequence, location of the `spikes.spk` file (see below), of the `retina.xml` file (see below), center of the retina in the input image, etc.

retina.xml is the file containing all characteristics of the retina used for the simulation. It has the same format as the input definition file (also in xml, see Section 3), except that it also contains the *positions and indexes of all ganglion cells created in the retina*, a necessary information to exploit the output spikes.

spikes.spk is an ASCII file containing all spikes emitted during the simulation, as emitted by library *MvaSpike*. All spikes emitted by the retina are saved in a single file, ranked according to their emission time:

```
9128  0.0153717
9272  0.015372
7900  0.0153743
9332  0.015375
7692  0.0153755
etc.
```

The first number is an absolute index for the cell that emitted the spike. It can be linked to a spatial position thanks to the `retina.xml` file. Second number is the time of the spike. Alternatively, the emitted spikes can be retrieved in the binary hdf5 format, by using option `-hdf5` when calling the executable.

Other output files are produced only if extra saving options are activated. Option `-savemap` saves all maps for intermediate signals (OPL current, bipolar cells, amacrine feedback, etc.), in specific directories termed `oplFrames/`, `bipolarFrames/`, etc. Option `-saveCP` only saves temporal courses for these signals at Center Pixel of the retina, in the form of 1-dimensional .inr images: `oplCP.inr`, etc. For both saving procedures, option `-nS` fixes the temporal frequency of saves, every `-nS` retinal time step.

2.2 The `TestGanglionCell` executable

This is a wrapper for program `Retina`, in the particular case where you want to test a *single*, spiking ganglion cell with noise in its spike generation, and reconstruct an averaged firing rate. This program can serve as a basis for simulating 'physiological' recordings on our model ganglion cells. Examples of use for this program can be found in directory `VirtualRetina/experiments/`

Most important options:

- `-tr` fixes the number of trials used to reproduce a firing rate for the cell. It is meaningful only if you defined some intrinsic noise in the spike generation process for the cell, in the retina definition file. If you set `-tr 0`, the cell does not fire spikes at all (only produces an analogical output).
- `-nodisp` if you want no display (useful when automatically testing the program over many cells and/or stimuli). By default, the program displays the average firing rate over the trials (output of the program), and also displays two intermediate signals: OPL current and ganglion input current. See Section 3, or better yet report [1], for significance of these intermediate signals.
- `-p` and `-lat` allow you to change the linear kernel used to reconstruct the average firing rate from the trains of spikes.
- etc. Full list of options by typing `-h` when calling the executable. In particular, most options for program `Retina` also work for program `TestGanglionCell`.

2.3 The `ReconstructRetina` executable

It allows to visualize the spiking output of the retina, by reconstructing a video sequence based on the spike trains emitted by program `Retina`. The reconstruction process is a simple linear summation:

Each emitted spike linearly contributes to the reconstruction by adding a spot, at the location of the emitting cell, for a particular length of time, expressed as a number of frames by option `-lat`. Spatially, the spot is a circle whose radius follows the log-polar scheme of the retina. Radius of a spot in the fovea, in pixels, is fixed with option `-w`. This defines the size of spots everywhere else, according to the log-polar scheme of the retina.

So, in three dimensions, the spot has a certain volume VolSpot expressed in voxels (that depends on the radial position of the emitting cell). The intensity of the spot in each voxel is then chosen as

$$(\text{VolSpot}.d(r)^2)^{-1},$$

where $d(r)^2$ is the local 2d density of cells in the region around the emitting cell (see equations (1) and (18) further on). It is important to normalize our spot by $d(r)$, so that in the end, the number stocked in each pixel of the reconstructed sequence has the dimension of an *average firing rate per spiking cell*.

If you do not want the intensity of spots to be divided by cell density $d(r)^2$, write option `-nodn`. In which case the number stocked in each pixel of the reconstructed sequence has the dimension of an average firing rate *per pixel*, and you see nothing in peripheral regions where there are less cells...

Other important options:

- `-i path/to/simulation.txt` gives the path to the simulation file emitted by program `Retina`, that you want to reconstruct from.
- `-ch` fixes which layers of ganglion cells you want to represent (all ON channels, for example). It is used as follows: `-ch 2 0 3` means that you want to represent two channels, namely channels number 0 and 3. So if you had a single layer in your retina (most of the time!), you still have to write `-ch 1 0`. Sorry...
- etc. Full list of options by typing `-h` when calling the executable.

2.4 Other executables

These are other executables (mostly, small utilities) I wrote for my work, included here in case they could help someone. Use `-h` on each of them for more precisions.

`viewVideo` is a simple video viewer; useful to watch input sequences, or reconstructed sequences. You might also want to use `inrcast`, a much more complete application included in the CImg library, that is included in `VirtualRetina/local/bin`. But unlike `inrcast`, `viewVideo` allows you to play the video (and at any speed you like, by using option `-s`).

`lumImage` allows you to change the mean luminosity and contrast of a sequence before feeding it to the retina.

`Grating` creates a moving bar stimulus. Ah! Ah! I'm so funny! Okay: it creates a *grating* stimulus, and nothing more.

`shapleyVictor` is a more experimental program, to test the responses of a single ganglion cell, with contrast-gain-control, to Shapley-Victor multi-sinus stimuli at different contrasts (reproduction of the first experiment of article 'Shapley-Victor 78'). See example of use in `../experiments/multi-sinus_shapley-victor78/`.

3 Writing a retina definition file in xml

All retinal parameters, for a given simulation, are given to program `Retina` in the form of a single retina definition file, hierarchically ordered in an xml structure. Some customized files are provided, and described in Section 5; but for full usage of the software, the structure of the xml file must be understood, as well as the underlying retinal model decribed in [1].

3.1 General architecture of a file

The retina is described through a hierarchical xml structure, whose father element is simply termed <`retina/`>. As in every xml tree, an *element* has children *attributes* in the form `param-name="value"`, as well as children elements. The attributes correspond to *numerical parameters* of the model. The children elements allow to hierarchically define sub-structures, which themselves possess parameters in the form of attributes. This allows:

1. clearer organisation of parameters for the retina, according to their location in the retinal model.
2. extended modularity of the retina, because sub-elements of node <`retina/`> are bricks that can be present or not in the architecture, making the resulting retinal model more or less complex.

Here is the general structure of the retina file, when only element <`retina/`> and its direct children are represented:

```
<retina  temporal-step__sec ="0.005"
         input-luminosity-range="255"
         pixels-per-degree="10.0">

         <log-polar-scheme/>
         <outer-plexiform-layer/>
         <contrast-gain-control/>
         <ganglion-layer/>
         . . .
         <ganglion-layer/>

</retina>
```

The three first lines show attributes to element <`retina/`>. They are general parameters that concern the whole retinal scheme:

`temporal-step__sec` is the (simulated) time length of one discretization step for the retina. Note that this is not the precision of emitted spike trains! Rather, the spike trains are generated from a current that is constant within each bin of 5 ms (in this case).

`input-luminosity-range` is the intensity corresponding to color white in the input sequences fed to the retina. This allows not to change all subsequent amplitude parameters whenever the retina is fed images with another luminosity encoding.

`pixels-per-degree` is, in the same spirit, a conversion factor allowing to express all subsequent spatial scales of the retina in terms of visual degrees,

rather than pixels. Which allows to change a single parameter (this one) when one wants to change how close the input image is to the simulated eye.

Note that, whenever a parameter is expressed in a unit that is not trivial, this unit is expressed in the xml name for the parameter, after a double underscore.

The following lines correspond to specific, independent sub-elements of the underlying retina model, as detailed in [1]. Most of them are *optional* features of the retina: only element <`outer-plexiform-layer/`> is always necessary. We detail these elements in the sequel, but here is a rapid overview of their signification:

<`log-polar-scheme/`> deals with the spatially non-uniform structure of the retina. If this node is absent, then the retina is taken to have a uniform density of cells. If this node is present, its parameters define a central *fovea* with uniform density of cells, and then a radially decreasing density.

<`outer-plexiform-layer/`> is where the *center-surround* architecture of the retina is defined. It is the only block that must always be present in our retinal definition file. However, this stage is available in different versions with more or less parameters, as detailed in the sequel.

<`contrast-gain-control/`> defines parameters for an optional contrast gain control feedback, as modeled in [1].

<`ganglion-layer/`> models one layer of ganglion cells, that can be X-type or Y-type, and have ON or OFF polarity. There can be as many of these layers as one desires, plausibly with different filtering properties.

Let us now review these elements more in detail. All the associated mathematical formulas are explained more thoroughly in [1].

3.2 Log-polar scheme

When this element is present, it writes:

```
<log-polar-scheme  fovea-radius__deg="1.0"
                   scaling-factor-outside-fovea__inv-deg="1.0"/>
```

These two parameters, let's term them respectively R_0 and K, define a scaling function throughout the whole retina by:

$$s(r) = \begin{cases} 1 & \text{if } r < R_0, \\ \big(1 + K(r - R_0)\big)^{-1} & \text{if } r > R_0, \end{cases} \tag{1}$$

where r is the eccentricity from the center of the retina, measured in retinal degrees. At a given eccentricity r, all spatial scales of filtering in the model are proportional to $s(r)^{-1}$. Similarly, the local density of ganglion cells at this location is proportional to $s(r)^2$. If no <`log-polar-scheme/`> is present, then $s(r)$ is taken as constant in the whole retina (uniform retina).

3.3 Outer Plexiform Layer

This stage is the basis of retinal processing; it is where the *center-surround* architecture of the retina arises. Element <`outer-plexiform-layer/`> is always present, but it can exist in different versions.

3.3.1 Linear Version

The so-called 'linear version' of the OPL stage writes:

```
<outer-plexiform-layer>
     <linear-version
          center-sigma__deg="0.03"
          surround-sigma__deg="0.1"
          center-tau__sec="0.01"
          surround-tau__sec="0.01"
          opl-amplification="10"
          opl-relative-weight="1"
          leaky-heat-equation="1" />
</outer-plexiform-layer>
```

It implements the following linear filtering on the input image:

$$I_{\mathrm{CS}}(x,y,t) = \lambda_{\mathrm{OPL}}\big(C(x,y,t) - wS(x,y,t)\big), \tag{2}$$

with

$$C(x,y,t) = K_{\sigma_C,\tau_C} * L \ \ (x,y,t), \tag{3}$$
$$S(x,y,t) = K_{\sigma_S,\tau_S} * C \ \ (x,y,t), \tag{4}$$

where C denotes *center* signal, and S denotes *surround* signal. Sign $*$ represents spatio-temporal convolution. As explained in [1], this filter is a *band-pass* filter that enhances spatial edges and temporal changes. It arises due to the interaction of light receptors and horizontal cells in the retina.

$L(x,y,t)$ is the input luminosity profile, divided by its maximum possible value `input-luminosity-range` (see Section 3.1).

λ_{OPL} is the overall gain of the center-surround filter, fixed by attribute `opl-amplification`. It is chosen so that $I_{\mathrm{CS}}(x,y,t)$ (or $I_{\mathrm{OPL}}(x,y,t)$, when one uses the *adapting version* of the OPL presented afterward) be a dimensionless signal, with magnitudes of the order of unity.

w in equation (2) is the relative weight of *center* and *surround* signals. The number is between 0 and 1, physiologically close to 1. For $w = 1$, the OPL filter is totally band-pass. It is fixed by attribute `opl-relative-weight.`

$K_{\sigma,\tau}$ is a positive spatio-temporal low-pass filter, of integral one, defined by

$$K_{\sigma,\tau}(x,y,t) = G_\sigma(x,y)\exp(-t/\tau)/\tau, \tag{5}$$

if $t > 0$, and zero otherwise. $G_\sigma(x,y)$ is the two-dimensional normalized Gaussian distribution of standard deviation σ.

Parameters σ_C and τ_C are fixed by respective attributes `center-sigma__deg` and `center-tau__sec`. Parameters σ_S and τ_S are fixed by respective attributes `surround-sigma__deg` and `suround-tau__sec`.

For both σ_C and σ_S, values are fixed *in the fovea*. Outside of the fovea, at eccentricity r, the scales of filtering are given by the scaling factor $s(r)$ in (1), and the formula

$$\sigma(r) = s(r)^{-1}\sigma_{\mathrm{fovea}}. \tag{6}$$

This is also true for other spatial scales of filtering in the sequel.

NOTA: If attribute `leaky-heat-equation` is set to 1 rather than 0, the expression for filters $K_{\sigma,\tau}$ becomes slightly different. One has:

$$K_{\sigma,\tau}(x,y,t) = G_{\sigma\sqrt{t/\tau}}(x,y)\exp(-t/\tau)/\tau, \tag{7}$$

where $G_\sigma(x,y)$ is again a normalized Gaussian. This filter is slightly more plausible biologically and slightly faster, but the overall difference with (5) is slight. See [1] for details on this equation.

3.3.2 Undershoot version

The so-called 'undershoot version' of the OPL is *also* a linear filter, but with an additional temporal high-pass stage that models *slow transient* properties of retinal filtering. The undershoot version writes:

```
<outer-plexiform-layer>
     <undershoot-version
          center-sigma__deg="0.03"
          surround-sigma__deg="0.1"
          center-tau__sec="0.01"
          surround-tau__sec="0.01"
          opl-amplification="10"
          opl-relative-weight="1"
          leaky-heat-equation="1"

          undershoot-relative-weight="0.5"
          undershoot-tau__sec="0.2"
          />
</outer-plexiform-layer>
```

The supplementary transient writes:

$$I_{\mathrm{OPL}}(x,y,t) = K_U \overset{t}{*} I_{\mathrm{CS}}\ \ (x,y,t), \tag{8}$$

where $I_{CS}(x,y,t)$ is obtained as before (equation (2)). $\overset{t}{*}$ denotes temporal convolution, and $K^{adap}(t)$ is a partially high-pass temporal filter defined by

$$K_U(t) = \delta_0(t) - w_U \exp(-t/\tau_U)/\tau_U, \tag{9}$$

where $\delta_0(t)$ is a Dirac function, representing the original signal. w_U is a constant between 0 and 1 giving the relative strength of the adaptation effect, fixed by xml attribute `undershoot-relative-weight`. It was chosen to fit experimental data, with typical values of around 0.7, a value compatible with measurements of light-evoked responses in retinal cones. τ_U is the temporal scale of the cellular adaptation, fixed by attribute `undershoot-tau__sec`, typically around 100 ms.

The OPL filter proposed in [1] is precisely this `adaptation-version.` The adaptation scheme allows better reproduction of experimental curves; however, using the `linear-version` instead is probably sufficient for most applications.

3.3.3 Luminance gain control version

One last version for the OPL has been impemented, modeling the shunting (divisive) influence of horizontal cells on light receptors. This is an equivalent, in the OPL, of the contrast gain control scheme proposed for the IPL in [1], which is explained afterwards in this tutorial.

This version is yet very experimental, so we will not detail it for the moment; just note that it exists, and will probably be studied and 'officialised' in a close future. For the moment, the `luminance-gain-control-version` is defined as:

```
<outer-plexiform-layer>
     <luminance-gain-control-version
          input-amplification-phototransduction="100"

          sigma-receptors__deg="0.03"
          sigma-horizontal-cells__deg="0.1"
          tau-phototransduction__sec="0.01"
          additional-tau-horizontal-cells__sec="0.01"

          inert-leak-in-receptors="10"
          horizontal-feedback-amplification="100"
          horizontal-feedback-nernst-potential="0"
          relative-weight-surround-center="1"
          />
</outer-plexiform-layer>
```

3.4 Contrast gain control in bipolar cells

Contrast gain control is the stage of *Virtual Retina* that aims at reproducing the non-linear changes in retinal filtering due to the ambient contrast in the scene. We model it as a shunting feedback from a certain type of membrane conductances onto bipolar cells. In our retina definition file, the associated xml element writes:

```
<contrast-gain-control
          opl-amplification__Hz="150"
          bipolar-inert-leaks__Hz="5"
          adaptation-sigma__deg="0.5"
          adaptation-tau__sec="0.03"
          adaptation-feedback-amplification__Hz="100"/>
```

The underlying equations are the following:

$$\frac{dV_{\mathrm{Bip}}}{dt}(x,y,t) = \lambda'_{\mathrm{OPL}} I_{\mathrm{OPL}}(x,y,t) - g_{\mathrm{A}}(x,y,t) V_{\mathrm{Bip}}(x,y,t) \tag{10}$$

$$g_{\mathrm{A}}(x,y,t) = K_{\sigma_{\mathrm{A}},\tau_{\mathrm{A}}} * Q(V_{\mathrm{Bip}}) \;\; (x,y,t). \tag{11}$$

Here $g_{\mathrm{A}}(x,y,t)$ represents the summed effects of all leak conductances in the bipolar cell layer (including inert leaks). In our model, this total leak depends on recepnt values of bipolar cells, through a feedback. $K_{\sigma_{\mathrm{A}},\tau_{\mathrm{A}}}(x,y,t)$ is a coupling low-pass filter as in (5), corresponding to spatio-temporal diffusion of the

contrast signal, and $Q(V_{\mathrm{Bip}}(x,y,t))$ is the synaptic current induced by bipolar cells in amacrine cells, chosen to be a quadratic function (see [1]):

$$Q(V_{\mathrm{Bip}}) = g_{\mathrm{A}}^0 + \lambda_{\mathrm{A}} V_{\mathrm{Bip}}^2. \tag{12}$$

Attribute `opl-amplification__Hz` fixes λ'_{OPL} in (10), turning the dimensionless signal $I_{\mathrm{OPL}}(x,y,t)$ in (8) into a signal with the dimension of a frequency. Parameter g_{A}^0 is fixed by `bipolar-inert-leaks__Hz`, while λ_{A} is fixed by `adaptation-feedback-amplification__Hz`. Other attributes have explicit names. Spatial filtering scale σ_{A} is defined in the fovea. Elsewhere it follows the scaling function, as in (6).

NOTA: Amplification factor λ'_{OPL} can appear useless, since it is applied right after another linear factor, that is λ_{OPL} in (2). We split the 'opl to bipolar' amplification factor in two numbers, to keep the *brick-like* organization of the software: the output of the OPL stage must have magnitudes the order of unity, the output of the control gain control stage as well. This way, the contrast gain control stage can be removed or added without having to significantly change other parameters of the model.

3.5 Ganglion layers and spike generation

Ganglion cells are the last stage in *Virtual Retina*. By opposition with the preceding stages, this layer can be multiplied in different versions, according to the type of output cells desired: one can ask at the same time for ON and OFF cells, for primate parasol or midget cells, or for cat X or Y cells, without having to do several times the preceding retinal processing (OPL and contrast gain control). One layer of ganglion cells is associated to the general xml structure:

```
<ganglion-layer

          sign="-1"
          transient-tau__sec="0.03"
          transient-relative-weight="0.75"
          bipolar-linear-threshold="0"
          value-at-linear-threshold__Hz="100"
          bipolar-amplification__Hz="300"
          sigma-pool__deg="0" >

          <spiking-channel>
              <circular-spiking-channel
                 diameter__deg="10" fovea-density__inv-deg="1"
                 g-leak__Hz="50" sigma-V="0.0" refr-mean__sec="0.003"
                 refr-stdev__sec="0" random-init="1"/>
          </spiking-channel>

</ganglion-layer>
```

The attributes of element <`ganglion-layer/`> deal with further signal processing at the level of ganglion cells, still modeled as continuous maps. By opposition, subelement <`spiking-channel/`> explicitly defines an array of spiking cells, with its own set of parameters. If this spiking element is present, the

output of the retina is the set of emitted spike trains. If it is absent, the output of the retina is the continuous signal $I_{\mathrm{Gang}}(x, y, t)$ (equation (15)), that would otherwise be used to generate the spike trains, and can be considered as an average firing rate.

3.5.1 Further signal processing in ganglion cells

Attribute `sign` gives the polarity of the layer. 1 for ON cells, and -1 for OFF cells.

The two following attributes, `transient-tau__sec` and `transient-relative-weight`, define parameters for a linear high-pass stage modeling *fast transients* in the IPL (mostly due to amacrine cells):

$$V_{\mathrm{Bip}}^{trs}(x, y, t) = K^{trs} \overset{t}{*} V_{\mathrm{Bip}} \ \ (x, y, t), \tag{13}$$

where $\overset{t}{*}$ denotes temporal convolution, and $K^{trs}(t)$ (trs standing for *transient*) is a partially high-pass temporal filter, defined exactly as the OPL adaptation filter K_U in (9):

$$K^{trs}(t) = \delta_0(t) - w_G \exp(-t/\tau_G)/\tau_G, \tag{14}$$

where $\delta_0(t)$ is a Dirac function, representing the original signal, and w_G is a constant between 0 and 1.

This high-pass stage can be seen as modeling the inhibitory reciprocal connections onto bipolar cells from specific amacrine cells, at the level of bipolar cells' synaptic terminals in the IPL. It was found mandatory to reproduce the outputs of all types of ganglion cells. However the precise values of the parameters depend on the type of cell modeled (see Section 4 and [1]).

The next three attributes define the synaptic pooling and rectification from bipolar cells to ganglion cells.

$$I_{\mathrm{Gang}}(x, y, t) = N(V_{\mathrm{Bip}}^{trs}) \ (x, y, t), \tag{15}$$

where transmission function N in (15) is a smooth synaptic rectification defined by

$$N(v) = \begin{cases} \dfrac{T_0^2}{T_0 - \lambda_{\mathrm{BG}}(v - V_{\mathrm{BG}})} & \text{if } v < V_{\mathrm{BG}}, \\ T_0 + \lambda_{BG}(v - V_{\mathrm{BG}}) & \text{if } v > V_{\mathrm{BG}}. \end{cases} \tag{16}$$

V_{BG} is the 'linearity threshold' of the cell, i.e. the value after which transmission becomes linear. It is fixed by `bipolar-linear-threshold`. Note that $N(V_{\mathrm{BG}}) = T_0$. T_0 is fixed by `value-at-linear-threshold__Hz`, and λ_{BG} by `bipolar-amplification__Hz`.

Finally, parameter `sigma-pool__deg` defines a possible *post-synaptic pooling*, necessary to model primate parasol cells and cat Y cells (see Section 4 and [1]). The given value is for the fovea. Else, the width of the pooling follows the scaling function, as in (6).

3.5.2 Spike generation in ganglion cells

The spiking array associated to the layer of ganglion cells, when it is present, can be in two versions.

Square, uniform array of cells

```
<spiking-channel>
    <square-spiking-channel
          size-x__deg="10" size-y__deg="7.5" uniform-density__inv-deg="10"
          g-leak__Hz="50" sigma-V="0" refr-mean__sec="0.003"
          refr-stdev__sec="0" random-init="1" />
</spiking-channel>
```

`size-x__deg` and `size-y__deg` are the size of the array (in retinal degrees). Similarly, `uniform-density__inv-deg` sets the number of cells per retinal degree (1-dimensional density).

The four parameters on the following line are the spiking parameters. Each cell in the array is modelled as a simple LIF neuron, with possibly a realistic additional noise in the spike generation process. If C_n is a cell located at position (x_n, y_n) in the array, we generate its spiking output by:

$$\begin{cases} \dfrac{dV}{dt} = I_{\text{Gang}}(x_n, y_n, t) - g^L V(t) + \eta_v(t), & (17) \\ \text{Spike when threshold is reached: } V(t_{\text{spk}}) = 1, \\ \text{Refractory period: } V(t) = 0 \text{ while } t < t_{\text{spk}} + \eta_{\text{refr}}, \\ \text{and (17) again,} \end{cases}$$

where $\eta_v(t)$ and η_{refr} are the two noise sources that can be added to the process. $\eta_v(t)$ is taken as a Brownian movement that has the dimension of a current. Integration of this current through equation (17) is equivalent to adding to $V(t)$ a Gaussian auto-correlated process with time constant $1/g^L$ (typically, 20 ms), and variance σ_v (fixed by parameter `sigma-V`). The amplitude of $\eta_v(t)$ is chosen for σ_v to be around 0.1. η_{refr} is a stochastic absolute refractory period that is randomly chosen after each spike, following a normal law, typically $\mathcal{N}$(3 ms,1 ms).

Finally, parameter `random-init` decides whether the cells' potentials should be randomly initialized at the beginning of the simulation (to avoid artificial synchronies at image onset).
NOTA: Do not use this square array when you wish to simulate a single spiking cell, because the single constructed cell will not be at the very center of the image. Rather use the circular array of cells (next paragraph).

Circular array of cells

In case a log-polar scheme has been chosen for the retina (Section 3.2), one might prefer an array of spiking cells whose density respects the log-polar scheme. The circular array of cells is designed for this purpose:

```
<spiking-channel>
    <circular-spiking-channel
          diameter__deg="30" fovea-density__inv-deg="5"
          g-leak__Hz="50" sigma-V="0" refr-mean__sec="0.003"
          refr-stdev__sec="0" random-init="1" />
</spiking-channel>
```

The spiking procedure is similar to the uniform case. Only difference is that the size of the array is now fixed by a single number `diameter__deg` (in retinal degrees), while the one-dimensional density of cells is given *in the fovea* through attribute `fovea-density__inv-deg`.

Outside of the fovea, the one-dimensional density of cells $d(r)$ is given by the scaling factor $s(r)$ in (1), and the formula

$$d(r) = d_{\text{fovea}} s(r). \tag{18}$$

This concludes the xml definition of the retina model *per se*. One last component can be present in the retina definition file, that is a basic microsaccade generator.

3.6 Microsaccade generation

Simulation of the retina can possibly include a simple reproduction of fixational microsaccades, applied to the input image before it is passed to retinal treatment. If one wishes to use this functionality, one must include the saccade generator in the definition file, outside of the <`retina/`> node:

```
<retina-description-file>

   <basic-microsaccade-generator
          pixels-per-degree="10.0"
          temporal-step__sec ="0.005"

          angular-noise__pi-radians="0.3"
          period-mean__sec="1"
          period-stdev__sec="0.3"
          amplitude-mean__deg="0.3"
          amplitude-stdev__deg="0.1"
          saccade-duration-mean__sec="0.03"
          saccade-duration-stdev__sec="0.01"/>

   <retina. . ./>

</retina-description-file>
```

It implements straight microsaccades around the fixation point, with random amplitude, duration, and inter-saccadic interval. For conveniency, this object also expresses temporal scales in seconds and spatial scales in retinal degrees, with two conversion factors `pixels-per-degree` and `temporal-step__sec`. Make sure they are the same as for your retina!

4 Customization with species and pathway

Some parameters of the model vary according to the type of ganglion cell modeled. For example, the sizes of receptive fields, and the global density of retinal cells, are very different in cat or rabbit retinas and in primates, that have a much greater density of cells in their foveas, with smaller (more precise) receptive fields.

Similarly, some ganglion cells, such as cat Y cells, or primate Parasol cells, are reknown to display a spatial non-linearity, probably due to their wide dendritic tree, as well to possess responses that are more transient temporally.

Finally, primate Midget cells, which possess very small receptive fields, do not display contrast gain control like most retinal cells do. Table 1 sums up the most important parameters of the model that will vary according to the type of species and pathway modeled.

Parameter	Cat X	Cat Y	Parasol	Midget
OPL stage				
`center-sigma__deg`	0.5	0.5	0.05 (fov)	0.05 (fov)
`surround-sigma__deg`	1.5	1.5	0.15 (fov)	0.15 (fov)
contrast gain control				
`adaptation-sigma__deg`	1.5	1.5	0.15 (fov)	0.15 (fov)
`adaptation-feedback-amplification__Hz`	100	100	100	0
`bipolar-inert-leaks__Hz`	5	5	5	50
ganglion layer				
`transient-relative-weight`	0.7	1	0.7-1	0.7
`bipolar-amplification__Hz`	100	400	100-400	100
`sigma-pool__deg`	0	1.5	0-0.15 (fov)	0 (fov)

Table 1: Model parameters that vary according to species and pathway.

5 Examples of use

We now propose examples of use of Virtual Retina. All command lines are written for execution from `Retina_Package/VirtualRetina`. We will test the program on the input sequences already proposed in the package: video sequence `walking_finland` and static image `rocks_2contrasts.pgm`. Both these sequences can be viewed thanks to the `viewVideo` executable. For instance

```
local/bin/viewVideo test/sequences/walking_finland/finland.10* -s 100
```

allows to watch sequence `walking_finland` at a rate of 100 ms per frame. If option `-s 0` is chosen, then the sequence can be viewed frame per frame, using PageUp and PageDown buttons. We start off with the simplest linear retina model available.

5.1 Spikeless, linear retina

File `EXAMPLE_linear_noSpikes.xml`, included in directory `test/retina_files`, contains the xml definition of a simple *center-surround* retinal filter:

```
<retina-description-file>
   <retina  temporal-step__sec="0.04"
            input-luminosity-range="255"
            pixels-per-degree="10.0">

            <outer-plexiform-layer>
               <linear-version
                  center-sigma__deg="0.03"
                  surround-sigma__deg="0.1"
                  center-tau__sec="0.01"
                  surround-tau__sec="0.01"
                  opl-amplification="10"
                  opl-relative-weight="1"
                  leaky-heat-equation="1" />
            </outer-plexiform-layer>
   </retina>
</retina-description-file>
```

Only the OPL filter is present, in its simplest version. The two spatial parameters of filtering are tuned to a rough approximation of a primate fovea. The conversion to image pixels is done through parameter `pixels-per-degree`. Since this linear model does not display feedbacks, the retinal time step can be chosen quite large: here, 40 ms.

This linear retina can be tested with:

```
local/bin/Retina -ret test/retina_files/EXAMPLE_linear_noSpikes.xml
              test/sequences/walking_finland/finland.10* -r 1 -nS 1
```

Option `-r 1` asks that each input last for one retinal time step (hence, 40 ms). The output of this spikeless model is the map signal of the *last* layer present in the retina. Here, it happens to be current $I_{\mathrm{CS}}(x, y, t)$ (equation (2)), which

should have normalized values within the order of unity. By default, it is saved in directory `tmp/`, under the names `oplFrames/opl_signal_xxxxxx.inr`. One can check that this signal has indeed been saved (use PageUp and PageDown):

```
local/bin/viewVideo tmp/oplFrames/opl_signal*.inr -s 0
```

It displays an enhancement of image edges, and especially *temporal* edges: a strong activation is produced by the edges of the moving characters.

5.2 Single spiking cell

Another approach is to test the response of *one* precisely modeled ganglion cell, under classical physiological stimulations. File `test/retina_files/EXAMPLE_cat_X_cell.xml` defines the code for a single, spiking cat X ganglion cell, with contrast gain control:

```
<retina-description-file>

   <retina  temporal-step__sec="0.005"
            input-luminosity-range="255"
            pixels-per-degree="2.0">

            <outer-plexiform-layer>
               <undershoot-version
                  center-sigma__deg="0.88"
                  surround-sigma__deg="2.35"
                  center-tau__sec="0.01"
                  surround-tau__sec="0.01"
                  opl-amplification="10"
                  opl-relative-weight="1"
                  leaky-heat-equation="1"
                  adap-relative-weight="0.5"
                  adap-tau__sec="0.2"
                  adap-type="0" />
            </outer-plexiform-layer>

            <contrast-gain-control
                  opl-amplification__Hz="150"
                  bipolar-inert-leaks__Hz="5"
                  adaptation-sigma__deg="2.5"
                  adaptation-tau__sec="0.01"
                  adaptation-feedback-amplification__Hz="100"/>

            <ganglion-layer
                  sign = "1"
                  transient-tau__sec="0.03"
                  transient-relative-weight="0.7"
                  bipolar-linear-threshold="0"
                  value-at-linear-threshold__Hz="80"
                  bipolar-input-amplification__Hz="100">
```

```
                <spiking-channel>
                   <circular-spiking-channel
                    diameter__deg="1" fovea-density__inv-deg="1"
                    g-leak__Hz="50" sigma-V="0.1" refr-mean__sec="0.003"
                    refr-stdev__sec="0.001" random-init="1" />
                </spiking-channel>
            </ganglion-layer>
   </retina>
</retina-description-file>
```

Notice that the defined <`spiking-channel/`> contains a *single* ganglion cell (`diameter__deg` and `fovea-density__inv-deg` are both equal to `1`). Minimal sizes for the continuous retinal maps are automatically calculated, so as for side effects to be absent at the center of the maps, where the single ganglion cell is located.
NOTA: To simulate a single cell, use a *circular* spiking channel, which ensures that the single cell will be located exactly at the center of fixation of the retina.

One can test the response of this cell to a drifting grating:

```
local/bin/Grating -type 0 -f 4 -T 40 -Lum 255 -Cont 0.7 -o tmp/grating.inr
local/bin/TestGanglionCell tmp/grating.inr -r 1 -tr 10
                  -ret test/retina_files/EXAMPLE_cat_X_cell.xml
```

which creates a drifting grating (frequency 4 Hz, spatial period 40 pixels), and tests the defined ganglion cell on it over 10 trials, with executable `TestGanglionCell` (that is called with similar options as `Retina`).

One can also test the contrast gain control properties of the cell on a multi-sinus stimulus (see detailed explanations in [1]):

```
local/bin/shapleyVictor -T 10 -tr 0 -nP 10
                        -ret test/retina_files/EXAMPLE_cat_X_cell.xml
```

For more details on physiological recordings, see directory `experiments/`. For the multi-sinus experient, also try option `-h` on executable `shapleyVictor`.

5.3 Spiking retina with contrast gain control

We finish this tutorial with the presentation of file `EXAMPLE_primate_ParvoMagno.xml`, which presents a realistic array of spiking ganglion cells, emulating primate Parasol and Midget cells, both in their ON and OFF versions. This array possesses a radial structure with a fovea, non-linear contrast gain control[1], and spatial non-linearity for the Parasol cells.

We do not display the whole definition file in this tutorial: it is actually a bit indigest. Let us simply remark that this file appears very long because there are 4 ganglion layers, which are simple copies one of another, safe the following points.

[1] to display the perceptual interest of contrast gain control, we also display contrast gain control on our 'Midget' pathway; biologically however, primate Midget cells display few contrast gain control, as explained in Section 4.

- The two first ganglion layers are Midget ('Parvocellular') cells, whereas the two second are Parasol ('Magnocellular') cells. As a result, the two first layers do not display a spatial pooling `sigma-pool_deg` (if absent from the xml file, it is taken as zero by default); whereas the last two layers do display a spatial pooling. Similarly, values for attributes `transient-relative-weight` and `bipolar-amplification_Hz` are different in the two Midget layers than in the two Parasol layers.
- There are two ON layers (one Midget and one Parasol), and two OFF layers. Hence, between two layers of the same type (say Midget), the only difference is attribute `sign`, which is set at `1` for one layer, and at `-1` for the other.

We test the retina on the same sequence as precedingly:

```
local/bin/Retina -ret test/retina_files/EXAMPLE_primate_ParvoMagno.xml
            test/sequences/walking_finland/finland.10* -r 8 -nS 8
```

This time, since the retinal time step (defined in the retina xml file) is now of 5 ms only, we ask that each input frame last 8 retinal steps, that is 40 ms. Once the simulation is completed, one might wish to somehow view the emitted spike trains. One possibility is to use program `ReconstructRetina`:

```
local/bin/ReconstructRetina -i tmp/simulation.txt -ch 1 0 -f 2
                            -w 3 -o tmp/last_reconstruction.inr
local/bin/viewVideo tmp/last_reconstruction.inr -s 20
```

As explained precedingly, the `-ch` command fixes which ganglion layer(s) should be represented in the reconstruction from spikes. In this case, we asked for `1` channel of cells, channel number `0` (hence, the ON Midget cells). Other options have been explained in Section 2.3. Typing `-h` will provide more details.

References

[1] Adrien Wohrer, Pierre Kornprobst, and Thierry Viéville. Virtual retina: a biological retina model and simulator, with contrast gain control. Research Report 6243, INRIA, jul 2007.