Exploring the Diderot programming language and its applications to the visualization of neural models

Daniel Clark

Department of Electrical Engineering

December 18, 2013

Overview

- Review Diderot - uses, continuous fields, high-level simple syntax, parallelism
- Hypothesis: Can we use Diderot to define high-level brain operators (e.g. optical flow)?
- Neural field equation $\stackrel{?}{=}$ Diderot's continuous tensor field
- Initial experiments and limitations of Diderot
- Visualizing the phase plane with line integral convolution (LIC)
- Experimental results and future direction

Diderot's intended use

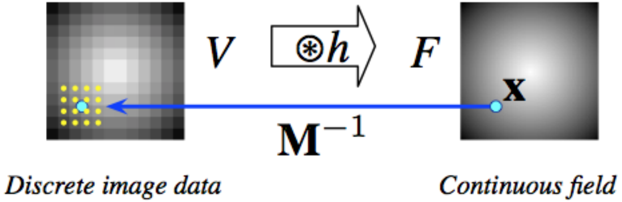
- Domain Specific Language (DSL) - designed for image analysis and visualization
- Image analysis - Extract *quantitative* geometric properties of objects from image data
- Image visualization - use image data in tandem with computer graphics to *qualitatively* describe image properties [1]
- Optimized for algorithms with large number of (mostly) independent computations
- Provides a high-level, simple and direct syntax

Language design and implementation

- Diderot is structured for dealing with data as continuous tensor fields
- Tensor field - scalars (order 0), vectors (order 1), matrices (order 2)
- Formed via convolution of discrete input with "continuous" kernel - `field#k(d)[σ] F = bspln3 ⊛ img`
- `k` - continuous derivatives, `d` - dimensionality, σ - tensor order
- Operators on field include -
  `t = F(pos), ∇F(pos), ∇⊗F(pos);`
  `F = ∇F, ∇⊗F, s*F,` F_1+F_2`;`
  `b = inside(pos,F);`
- `t` is a tensor, which can be manipulated by standard discrete operations, such as - $\bullet$, $\otimes$, $|\ |$, $*$

Field computation

Actual field computation is performed when field is probed -
`t = ∇F, ∇⊗F, F(pos);`



[2]

Field computation

e.g. `t = ∇F(pos);` This performs,

$$\begin{aligned}
\nabla F(x) &= (V \otimes \nabla h)(\mathbf{x}) \\
&= \left(V \otimes \begin{bmatrix} \frac{\partial}{\partial x} h \\ \frac{\partial}{\partial y} h \end{bmatrix} \right) \\
&= \begin{bmatrix} \sum_{i=1-s}^{s} \sum_{j=1-s}^{s} V[\mathbf{n} + < i, j >] h'(\mathbf{f}_x - i) h(\mathbf{f}_y - j) \\ \sum_{i=i-s}^{s} \sum_{j=1-s}^{s} V[\mathbf{n} + < i, j >] h(\mathbf{f}_x - i) h'(\mathbf{f}_y - j) \end{bmatrix},
\end{aligned} \tag{1}$$

where V - discrete input, h - (separable) continuous kernel, $\mathbf{x}$ - field position index, $\mathbf{M}^{-1}$ - space mapping matrix, $\mathbf{n} = \lfloor \mathbf{M}^{-1}\mathbf{x} \rfloor$ discrete mapped point, $\mathbf{f} = \mathbf{M}^{-1}\mathbf{x} - \mathbf{n}$

Program structure

- 3 sections - global definitions, strand, initialization
- Global - define *immutable* variables, inputs, load image data, usually create field
- Strand - computational core of application - run via parameters, init state variables (including output), `update`, `die`, `stabilize` methods
- A given strand will continue to execute its `update` method until the `stabilize` (write to output) or `die` (no write to output) method is called on that strand
- Initialization - define range of strand parameters (e.g. pixel indices)

Parallelism

- Bulk-synchronous parallelism model - super steps (`update` method), each consisting of asynchronous computations (individual `strand`s), executes until all strands `die` or `stabilize`
- Strands organized into (4096 supported) strands per block, followed by barrier synchronization at end of super step
- Parallel C code, OpenCL, CUDA (future)

Toy program

```
// ----- Global defs -----
int imgSizeX = 300;     // how many x pts user wants from field
int imgSizeY = 200;     // how many y pts user wants from field
int stepNum = 25;       // step limit before writing to output
image(2)[] img = load("../data/einstein.nrrd"); // import image
field#1(2)[] F = img  Ł  ctmr; // convolve img with kernel = field

// ----- Strand section -----
strand DEMO (int xi, int yi) {
    real xx = lerp(0.0, 3.0, -0.5, real(xi), real(imgSizeX)-0.5);
    real yy = lerp(0.0, 2.0, -0.5, real(yi), real(imgSizeY)-0.5);
    vec2 pos0 = [xx,yy];
    output real sum = F(pos0);
    int step = 0;
    // --- Update thread with math under condtions ---
    update {
        // Do some fancy math
        vec2 grad =   F (pos);    // take gradient of field and probe it
        sum = grad[0]+grad[1];
        step += 1;
        if (step == stepNum) {
          stabilize;            // write "sum" to output
        }
    }
}

// ----- Initialization section -----
initially [ DEMO(xi, yi) | yi in 0..(imgSizeY-1), xi in 0..(imgSizeX-1) ];
```

Hypotheses

1. We can define high-level operators (analogous to ∇) that will perform encoding of sensory data on a neural circuit.
2. We can represent this circuit as a continuous tensor field.

- Can we use Diderot as a convenient medium for accomplishing this?

Continuous neural fields

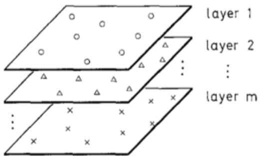
- Utilize maturely developed area of continuous dynamical systems to approximate neural circuitry
- Continuous equations - able to model neural activity as quantities in a continuous field, with functional relationships to sources and sinks *of that field* [3]
- Basic idea: activity of a neuron unit (e.g. average firing rate) in field layer i at point x, time t is

$$Z_i(x,t) = f_i\big(u_i(x,t)\big) \tag{2}$$

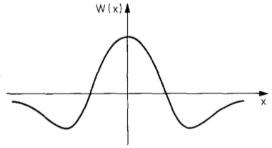
- $u_i(x,t)$ is average membrane potential, f_i is a non-linear activation function [4]

Continuous neural fields (cont'd)

- Model also assumes local excitatory and distant inhibitory inter-connectivity between neurons, described by time-varying weight function
- $w_{ij}(x, y; t)$ - influence activity from neuron at point y in layer j has on neuron at point x in layer i, t time units after firing initiates from neuron y



(a) Neural layers



(b) Weight function

Continuous neural fields (cont'd)

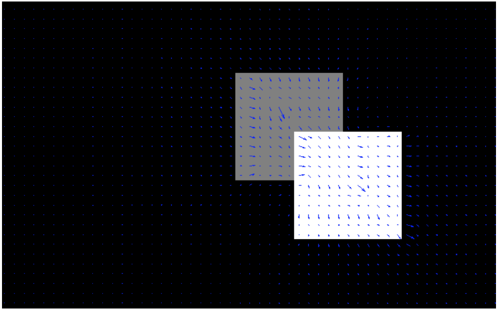
- Canonical continuous field equation [4]:

$$\tau_i \frac{\partial u_i(x,t)}{\partial t} = h_i + \Delta s_i(x,t) - u_i(x,t) + \sum_{j=1}^{m} \int_{\Omega_j} \int_{v} w_{ij}(x,y;t-v) Z_j(y,v)\ dv\ dy \tag{3}$$

- τ_i is the recovery time to steady-state, $\Delta s_i(x,t)$ is change in external stimulus, h_i is average distance to steady-state activity, Ω_j spatial neighborhood of summation on layer j
- Can think of neural field as $N_x \times N_t \times N_j$ continuous tensor field

Optical flow

- If we can define a neural circuit, such as in Eq. (3), what can we do with it?
- Put operation of continuous field into context - specific objective: encode optical flow
- Optical flow - quantification of relative motion in a vision sequence



Optical flow (cont'd)

- In general, three stages in detecting optical flow:
  1. Pre-filter/smooth input data spatiotemporally
  2. Extract relevant features (e.g. gradients)
  3. Weight and integrate features to produce field flow vectors
- Known, in general, as a differential or gradient-based approach

Horn-Schunk method

- Gradient constraint: change along the spatiotemporal path of a point of intensity is zero.

$$\frac{dI(\mathbf{x},t)}{dt} = 0 \;\Rightarrow \nabla_{\mathbf{x}} I(\mathbf{x},t) \cdot \mathbf{v} + \frac{\partial I(\mathbf{x},t)}{\partial t} = 0 \tag{4}$$

$$\nabla_{\mathbf{x}} I(\mathbf{x},t) = \begin{bmatrix} \frac{\partial I(\mathbf{x},t)}{\partial x} \\ \frac{\partial I(\mathbf{x},t)}{\partial y} \end{bmatrix}, \quad \mathbf{v} = \begin{bmatrix} u \\ v \end{bmatrix} \tag{5}$$

- Horn and Schunk posed this as an optimization problem with smoothness constraint, given by λ [5]

$$\min_{\mathbf{v}} \int_{\Omega} (\nabla I \cdot \mathbf{v} + \frac{\partial I}{\partial t})^2 + \lambda^2 (||\nabla u||_2^2 + ||\nabla v||_2^2) d\mathbf{x} \tag{6}$$

Horn-Schunk method (cont'd)

- HS iterative algorithm:

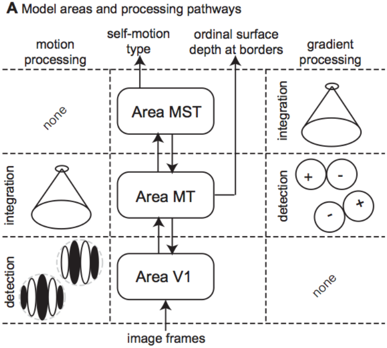
$$u^{k+1} = \bar{u}^k - \frac{I_x\left(I_x\bar{u}^k + I_y\bar{v}^k + I_t\right)}{\alpha^2 + I_x^2 + I_y^2} \quad (7)$$

$$v^{k+1} = \bar{v}^k - \frac{I_y\left(I_x\bar{u}^k + I_y\bar{v}^k + I_t\right)}{\alpha^2 + I_x^2 + I_y^2} \quad (8)$$

- $\bar{u}^k$, $\bar{v}^k$ are the average velocity values of kth iteration
- Can think of $\frac{dI}{dt}$ as $3 \times N_x \times N_y \times N_t$ continuous tensor field

Bio-inspired (Neumann) model

- $\frac{dI}{dt}$ gradients represented as likelihood values corresponding to membrane potentials of neurons
- Optical flow detection implemented as a three-level process:
  1. Spatiotemporal smoothing/feature extraction via three-stage cascade (V1)
  2. Integration of motion features (asymmetric filter kernels), detection for self-motion, feedback (MT)
  3. Integration of self-motion gradients, feedback (MST)



Bio-inspired (Neumann) model (cont'd)

- We'll focus on first level - three-stage cascade to implement:
- Feedfoward connectivity and temporal filtering

$$\dot{x}^{(1)} = -x^{(1)} + f_{sample}\left([x^{FF}]^{\alpha} * \Lambda_{space} \right) * \Lambda_{vel}. \tag{9}$$

- Feedback connectivity

$$\dot{x}^{(2)} = -x^{(2)} + x^{(1)}(1 + \beta x^{FB}). \tag{10}$$

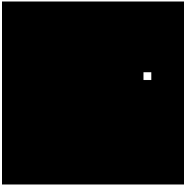
- Lateral (excitatory Λ^+, inhibitory Λ^-) connectivity

$$\dot{x}^{(3)} = -\gamma x^{(3)} + x^{(2)} * \Lambda^+ - x^{(3)}\left(x^{(2)} * \Lambda^-\right), \tag{11}$$

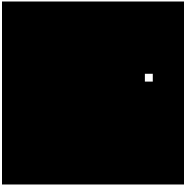
- Can think of likelihood values as $N_x \times N_y \times N_d \times N_v$ continuous tensor field

Gradient detection

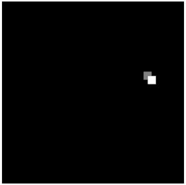
- Use binary image sequence for simplest case of optical flow detection



(a) 1st image



(b) 2nd image

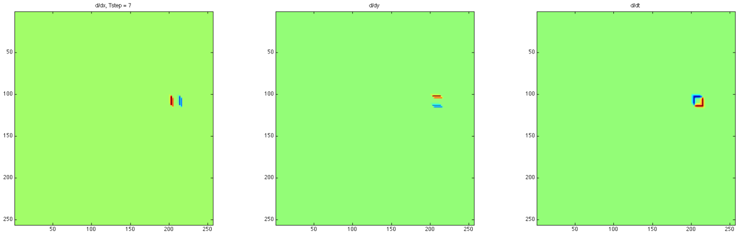


(c) 1st and 2nd

- Use Diderot to store this sequence as a continuous field F and use ∇ operator to verify we can calculate $\frac{dI}{dt}$

Gradient detection (cont'd)

- We implemented a simple function in Diderot to output the gradient of F, and plotted in MATLAB:



- We probed the field at 10 points temporally to get an interpolated flow between 1st and 2nd image

Limitations

- Next step: try iterative updates via Eq. (7) in Horn-Schunk method within Diderot
- Requires local averaging of gradients to yield $[\bar{u}^k, \bar{v}^k]^\top$
- Problem for Diderot - no convenient way to access neighboring points in F and perform average; can't use averaging kernel on a continuous field as it is defined in Diderot
- Even bigger limitation: no shared memory between strands; this prevents inter-neuron connectivity given in Eq. (3) from the neural field formulation, and Eq.(9), (10), and (11)
- Can we use Diderot for something else? $\rightarrow$ Image visualization

Harnessing Diderot

Recall novelties of Diderot:

1. Flexibility of dealing with "continuous" interpretation of discrete data (e.g. limited resolution)
2. Image visualization - high-level math operators common in visualization algorithms are conveniently built in
3. Parallel framework - algorithms that can be run in parallel can be executed quickly

Neuron dynamics and bifurcation

- Phase portrait - way to understand dynamics of conductance-based neuron models

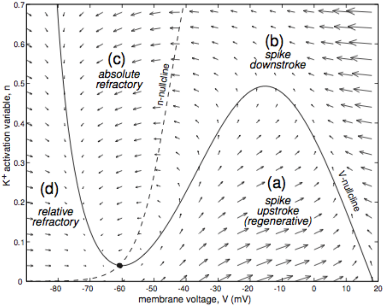


Figure : $I_{Na,p} + I_K$ model, with V and n nullclines [6]

- Compare three models of interest: Hodgkin-Huxley, Morris-Lecar, Fitzhugh-Nagumo

Reduced Hodgkin-Huxley model

Reduced-Hodgkin Huxley model (two-variable V, n):

$$C\dot{V} = I - g_L(V - E_L) - g_{Na}m_\infty(V)^3h_\infty(V)(V - E_{Na}) - g_K n^4(V - E_K)$$
$$\dot{n} = \frac{n_\infty(V) - n}{\tau_n(V)}$$

Morris-Lecar model

Morris-Lecar model:

$$C\dot{V} = I - g_L(V - E_L) - g_{Ca} m_\infty(V)(V - E_{Ca}) - g_K n(V - E_K)$$

$$\dot{n} = \frac{n_\infty(V) - n}{\tau_n(V)}$$

$$m_\infty(V) = \frac{1}{2}\left(1 + \tanh(\frac{V - V_1}{V_2})\right)$$

$$n_\infty(V) = \frac{1}{2}\left(1 + \tanh(\frac{V - V_3}{V_4})\right)$$

$$\tau_n(V) = \phi \cosh\left(\frac{V - V_3}{2V_4}\right)$$

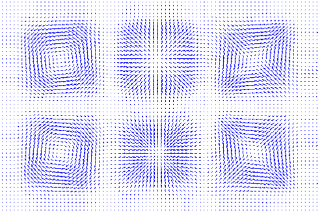
Fitzhugh-Nagumo model

Fitzhugh-Nagumo model:

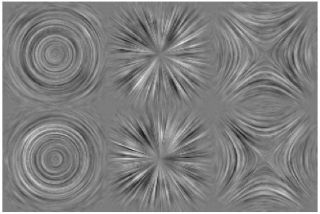
$$\dot{V} = V(a - V)(V - 1) - w + I$$
$$\dot{w} = bV - cw.$$

Line integral convolution

- Gradients figure is convenient for seeing general flow of states, but is distorted to show flow better
- Would be nicer to see this dynamic as a "continuous flow"
- Line integral convolution (LIC) - integrates underlying texture field F along gradient vectors in V to yield more intuitive image



(a) Original vector field



(b) LIC over white noise

Figure : LIC results from Diderot program *lic.diderot*

Line integral convolution (cont'd)

- In general, integrate some kernel function $\Lambda(w)$ over step s^k and step size Δs^k on kth iteration

$$h^k = \int_{s^k}^{s^k+\Delta s^k} \Lambda(w)dw$$

$$P^k = P^{k-1} + \frac{V\left(\lfloor P^{k-1} \rfloor\right)}{\left|\left|V\left(\lfloor P^{k-1} \rfloor\right)\right|\right|} \Delta s^{k-1}$$

$$LIC(x,y) = \frac{\sum_{k=0}^{l_f} F\left(\lfloor P_f^k \rfloor\right) h_f^k + \sum_{k=0}^{l_b} F\left(\lfloor P_b^k \rfloor\right) h_b^k}{\sum_{k=0}^{l_f} h_f^k + \sum_{k=0}^{l_b} h_b^k}$$

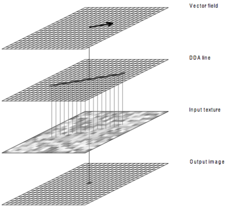
- F - underlying texture field to integrate over, $\lfloor P^k \rfloor$ - floored-position index, V - gradient vector field, h^k - kernel weight, l - number of iterations over k to perform

Line integral convolution (cont'd)

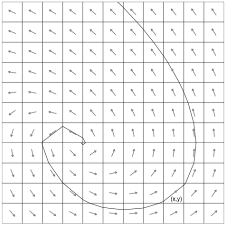
We can simplify the equations by using box kernel and fixed step-size Δs and step number l for forward and backward, with $P_0 = (x, y)$

$$P_f^k = P_f^{k-1} + \Delta s V(P_f^{k-1}), \quad P_b^k = P_b^{k-1} - \Delta s V(P_b^{k-1})$$

$$\Rightarrow LIC(x, y) = \frac{\sum_{k=0}^{2l} F(P_f^k) + F(P_b^k)}{2l + 1}$$



(a) Integration



(b) Stream line [7]

Generating initial phase portraits

- Generate gradient vectors via MATLAB's `quiver`

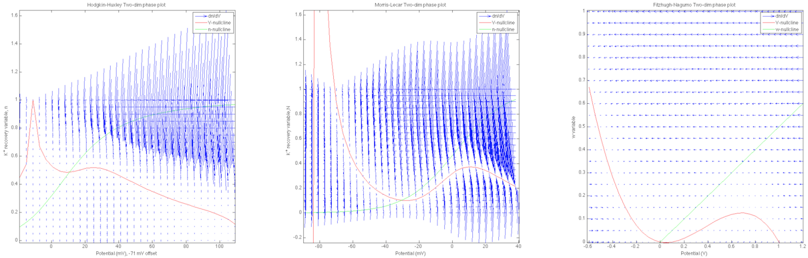


Figure : Un-normalized phase portraits for each neuron model

- Need a normalization to get a better result, without distorting image

Normalization and multi-path limit cycles

- Normalize gradients by a factor of input voltage range

$$dV_{norm} = dV/\eta\big(\max(V) - \min(V)\big). \tag{12}$$

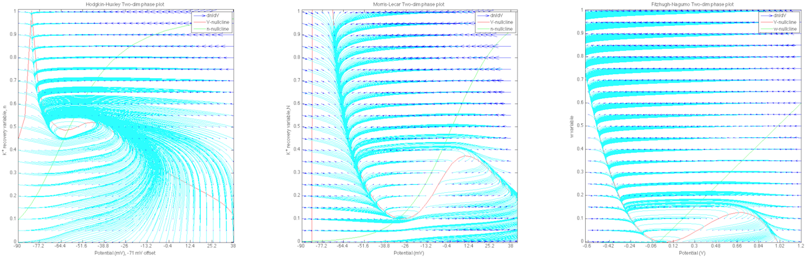
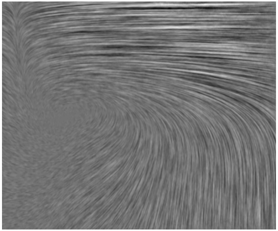


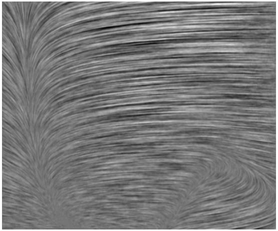
Figure : Normalized phase portraits for each neuron model

- Multi-path limit cycle runs verify the `quiver` vectors

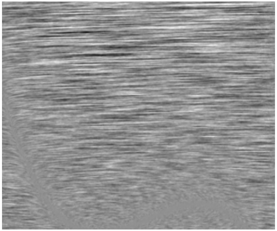
LIC of input phase portraits using Diderot



(a) Hodgkin-Huxley

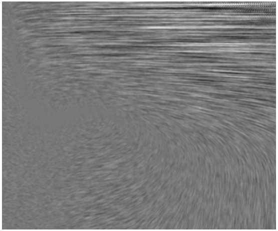


(b) Morris-Lecar

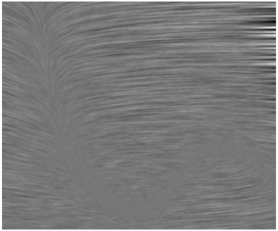


(c) Fitzhugh-Nagumo

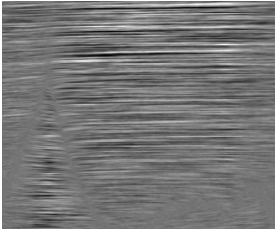
Generating phase portraits/LIC using Diderot (cont'd)



(a) Hodgkin-Huxley



(b) Morris-Lecar



(c) Fitzhugh-Nagumo

Generating phase portraits/LIC using Diderot (cont'd)

- Generating phase portrait/LIC entirely within Diderot was accurate with respect to the plots from MATLAB
- More convenient having results done all in one place
- More accurate using interpolation of $V, \ n$ input range rather than interpolation over vector gradients - functions of nonlinear diff eqs
- Extend this for time-varying injected current - demo videos...

Diderot benchmarking

	Sequential C	Parallel C	OpenCL	CUDA
2.7 GHz Core i5 iMac	30 m 43.8 s	9 m 20 s	Alloc. error	N/A
Huxley Cluster	94 m 39.6 sec	25 m 15 s	Alloc. error	N/A

Table : Benchmark for *licMLInject.diderot*

	Sequential C	Parallel C	OpenCL	CUDA
2.7 GHz Core i5 iMac	7.71 s	2.04 s	59.39 s	N/A
Huxley Cluster	9.59 s	2.71 s	0.85 s	N/A

Table : Benchmark for *licp.diderot*

*Huxley cluster did manage to utilize OpenCL backend to successfully compile and run (reduced size) *licHHInject.diderot*

Summary

- Diderot - great tool for interacting with data from a high-level domain
- Provides flexibility to manipulate data at user-specified granularity, regardless of input resolution
- (Mostly) independent image visualization/analysis algorithms best fit for use in Diderot
- However, not ideal at this time to implement high-level brain operators or perform any sort of sophisticated feature extraction on input sequences
- Despite this limitation, we were able to use Diderot as a visualization tool to bring about a convenient and intuitive way to understand the phase response of various neuron models

Future direction

- Three-dimensional LIC program for phase portrait over the V, n, m plane over Hodgkin-Huxley and Morris-Lecar models, would yield a more sophisticated depiction of the underlying model dynamics.
- LIC is inherently capable of supporting three-dimensional data, though how to visualize the output of the data in a meaningful way? Time-varying input current?
- Diderot's authors: implement inter-strand communication, global mutable memory, CUDA-supported backend, higher memory allocation for OpenCL
- With these functionalities, Diderot would be well equipped for further research in neural encoding and brain operators

Thanks

Special thanks to Yiyin Zhou and Nikul Ukani for directing my progress and giving feedback and suggestions on a daily basis. Thanks to Lev Givon for helping me with running experiments on the Huxley cluster. Thanks to Prof. Lazar for supervising the project. And thanks to the rest of the Bionet lab for various tips and consulting when I had questions.

References I

Gordon Kindlmann.
Diderot: A parallel dsl for image analysis and visualization.
http://youtu.be/m1cwdZL8mXk, October 2013.

Charisee Chiw, Gordan Kindlmann, John Reppy, Lamont Samuels, and Nick Seltzer.
Diderot: A parallel dsl for image analysis and visualization.
In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '12)*, pages 111–120, June 2012.

J.S. Griffith.
A field theory of neural nets: I: Derivation of field equations.
The bulletin of mathematical biophysics, 25(1):111–120, 1963.

References II

Shun-ichi Amari.
Dynamics of pattern formation in lateral-inhibition type neural fields.
Biological Cybernetics, 27(2):77–87, 1977.

J. L. Barron, D. J. Fleet, and S. S. Beauchemin.
Performance of optical flow techniques.
INTERNATIONAL JOURNAL OF COMPUTER VISION, 12:43–77, 1994.

E.M. Izhikevich.
Dynamical Systems in Neuroscience.
Computational neuroscience. MIT Press, 2007.

References III

Brian Cabral and Leith Casey Leedom.
Imaging vector fields using line integral convolution.
In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '93, pages 263–270, New York, NY, USA, 1993. ACM.