Curvature Analysis of Cardiac Excitation Wavefronts

A. Murthy[1], E. Bartocci[1,2], F. Fenton[3], J. Glimm[2], R. Gray[4], S.A. Smolka[1], and R. Grosu[1]

[1]Department of Computer Science, Stony Brook University, Stony Brook, NY
[2]Department of Applied Mathematics and Statistics, Stony Brook University, Stony Brook, NY
[3]Department of Biomedical Sciences, Cornell University, Ithaca, NY
[4]U.S. Food and Drug Administration, Silver Spring, MD

ABSTRACT

We present the *Spiral Classification Algorithm* (SCA), a fast and accurate algorithm for classifying electrical spiral waves and their associated breakup in cardiac tissues. The classification performed by SCA is an essential component of the detection and analysis of various cardiac arrhythmic disorders, including ventricular tachycardia and fibrillation. Given a digitized frame of a propagating wave, SCA constructs a highly accurate representation of the front and the back of the wave, piecewise interpolates this representation with cubic splines, and subjects the result to an accurate curvature analysis. This analysis is more comprehensive than methods based on spiral-tip tracking, as it considers the entire wave front and back. To increase the smoothness of the resulting symbolic representation, the SCA uses a weighted overlapping of adjacent segments which increases the smoothness at join points. To significantly speed up SCA computation time, we develop a GPU-CUDA implementation of SCA. SCA has been applied to several representative types of spiral waves, and for each type, a distinct curvature evolution in time (signature) has been identified. Moreover, distinguished signatures have been also identified for spiral breakup. This represents a significant first step in automatically determining parameter ranges for which a computational cardiac-cell network accurately reproduces ventricular fibrillation. The connection between parameters and physiological entities would then lead to an understanding of the root cause of the disorder and enable the development of personalized treatment strategies.

1. INTRODUCTION

An estimated number of eighty one million American adults, that is more than one in three adults, have one or more types of cardio-vascular disorders [7]. Among these disorders, ventricular tachycardia and especially fibrillation, may have devastating consequences (see Figure 1 and [3]).

Determining the physiological conditions responsible for a cardiac disorder is a grand challenge whose answer may lead

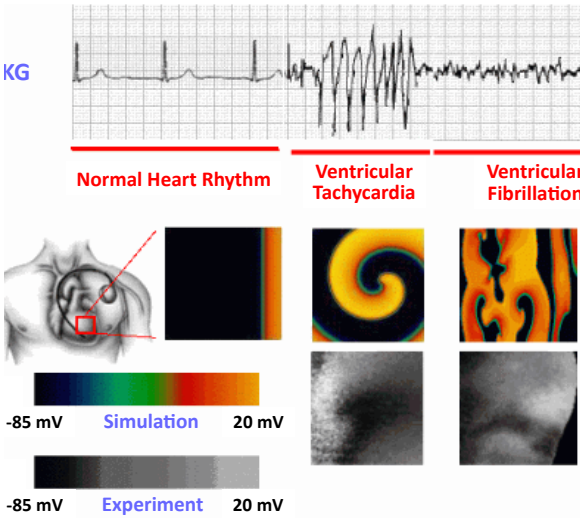


Figure 1: Emergent behavior in cardiac-cell networks. Top: Electrocardiogram. Middle and bottom: Simulation and experimental mappings of spiral waves of electrical activity occurring in the heart during tachycardia and fibrillation.

to personalized treatment strategies. An important aid to this quest is the mathematical modeling, analysis and simulation of cardiac-cell networks [2]. Among the myriad of existing mathematical models, differential-equation models of reaction-diffusion type (DEMs) are arguably the most popular. In the context of DEMs the above challenge can be reformulated as follows: *For what parameter ranges does a DEM network accurately reproduce the cardiac disorder?*

The past two decades have witnessed the development of increasingly sophisticated DEMs [4], ranging from 4 to 67 state variables and from 27 to 94 parameters [1, 10, 14, 16, 6]. The increase in the number of state variables reflects the technological advances in capturing the intrinsic ionic mechanisms more accurately. The increase also leads to a simplification of the differential equations. For example, most of the equations of the 67 variables DEM in [6] are of multi-affine type and were obtained using the law of mass action.

Unfortunately, the increase in the number of state variables, inevitably leads to an increase in simulation time. In particular, the simulation of the 67 variables DEM is so slow, that its authors only performed it in 1D (in a cable). In an accompanying paper, we present a CUDA-GPU implementation of the above-cited DEMs, on both Tesla and Fermi cards, with a dramatic reduction in simulation time. This

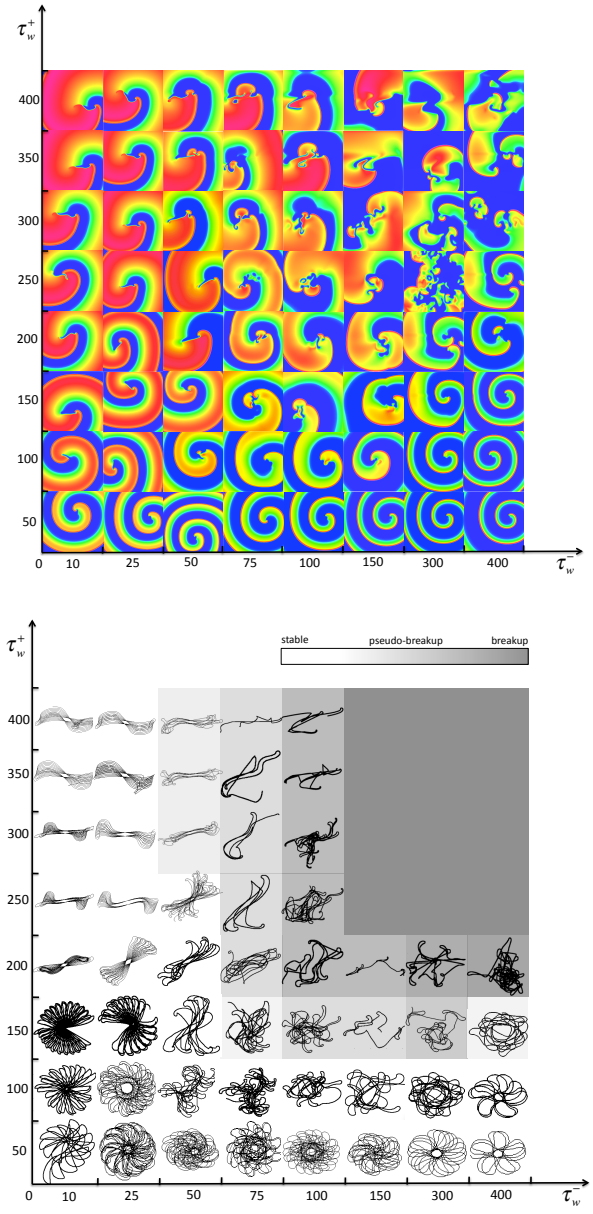


Figure 2: Parameters τ_w^-/τ_w^+: (a) Wave forms. (b) Tip movement and regions of fibrillation.

allows us to perform for the first time on a desktop a 2D (on a surface) simulation of the 67 variables DEM.

Despite of the difference in the number of state variables, the above DEMs are in some sense equivalent. Like in genetic regulatory networks (GRNs), the reduction in the number of variables can be understood as the result of a reaction (or time-scale) abstraction [13]. Akin to GRNs Michaelis-Menten or Hill-function abstractions, they reduce the number of state variables, at the expense of more complicated, sigmoidal dependencies among the remaining variables.

The lowest-dimensional DEM reproducing the experimentally mesoescopic behavior of cardiac cells with great accuracy, is the 4 variables minimal model (MDM) of co-author Fenton [1]. The CUDA-GPU implementation of the MDM is so fast, that it allows the real time simulation of an 500 by 500 grid of cells: One second of the MDM simulation time is approximately equal to one second of real time.

This increase in the MDM simulation speed, enables a systematic exploration of its associated parameter space. For example, in a CMACS (Computational Modeling and Analysis of Complex Systems) workshop, organized this Spring at the Lehman College, we used "crowd sourcing" (CS), to explore pairs of parameter ranges that lead to fibrillation: Given a spiral-initiation protocol, the students were asked to simulate the MDM on each node of a 2D grid of parameter values, capture the generated waves and track the tip movement of their spirals. The results obtained, for example for MDM parameters τ_w^-/τ_w^+, are shown in Figure 2.

CS provided very encouraging initial results. However, CS is not likely to scale up to the exploration of parameter spaces involving more than two parameters. Such exploration requires at least two ingredients: 1) A principled way of partitioning the parameter space, and 2) A fast and accurate algorithm for classifying spiral waves and their breakup.

In a recent paper [5], we present a hybrid-automaton (MHA) approximation of the MDM, which reproduces the MDM behavior with high accuracy. In each of its modes, the MHA is *multiaffine in the state variables* and *affine in the parameters*. This allows a bifurcation analysis of the MHA which results in a partitioning of its parameter space into behaviourally-equivalent hypercubes. Each face (hyperplane) of the hypercubes separates the positive and negative ranges of the derivative of one of the MHA state variables. Hence, for simulation and analysis purpose, it is enough to select one parameter vector from each hypercube.

In this paper we present a fast and accurate algorithm (SCA) for classifying spiral waves and their associated breakup. Given a digitized frame of a propagating wave, the SCA constructs a highly accurate representation of the front and the back of the wave, piecewise interpolates this representation with cubic splines, and subjects the result to an accurate curvature analysis. This analysis is more comprehensive than spiral-tip tracking, as it considers the entire front and back of a wave. To increase the smoothness of the resulting symbolic representation, the SCA uses a weighted overlapping of adjacent segments which increases the smoothness at join points. To speed up the SCA computation time, we employ a GPU-CUDA implementation. The SCA has been applied to several representative types of spiral waves generated with the MHA, and for each type a distinct curvature evolution in time (signature) has been identified. Moreover, distinguished signatures have been also identified for spiral breakup. This gives us hope that a *fully automatic answer to the grand challenge stated at the beginning of this section is within reach:* For what parameter ranges does an MDM network accurately reproduce ventricular fibrillation? Using the connection between parameters and physiological entities would then allow to understand the root cause of the disorder and develop personalized treatment strategies.

The rest of the paper is organized as follows. In Section 2 we discuss how to obtain the wavefront from a digitized frame. Section 3 discusses isopotential curvature estimation. Section 4 discusses our Bezier fitting algorithm which is improved in Section 5. Section 6 explains the symbolic evaluation of the curvature along the isopotentials. Finally,

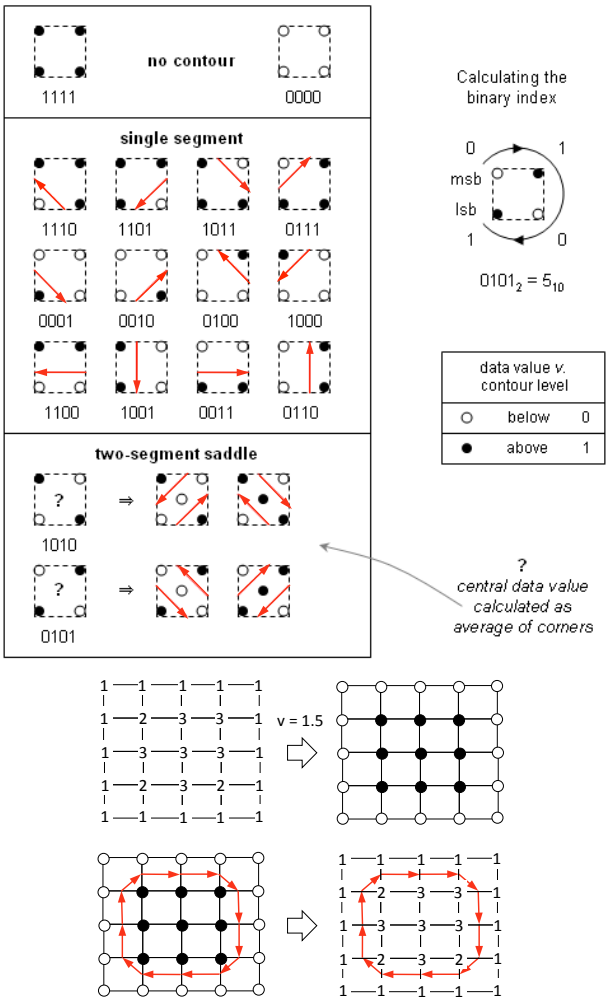


Figure 3: Marching squares.

Section 7 discusses our case studies results and directions of future work.

2. ISOPOTENTIAL RECONSTRUCTION

The simulation data obtained by executing an $N \times N$ grid of MHAs, or the experimental data obtained by optically mapping a cardiac tissue, with resolution $N \times N$, is a sequence of *digital frames* F_t of dimension $N \times N$. Each frame F_t is the snapshot at time t, with resolution $N \times N$, of the transmembrane *electrical potential* V, of the cardiac tissue.

The *wave-fronts (wave-backs)* of V_t, are the regions of V_t where each cell is in the activation (recovery) phase, that is their electrical potential $V_t(x, y)$ equals -30mV, and their derivative $dV(x, y)/dt$ is positive (negative). The two fronts define together the *isopotentials* I_{-30} of V_t. These are curves of constant voltage in V_t, that never intersect each other. For simplicity, when there is no danger of confusion, we will subsequently drop the subscript t occurring in F_t and V_t.

For any value v, the isopotentials I_v of V are smooth curves, whereas the isopotentials I_v of F are not. This is a consequence of the $N \times N$ resolution. The *isopotential reconstruction problem (IRP)* is therefore defined as follows: *Given a frame F, and a voltage value v, reconstruct the smooth I_v isopotentials of V.* As we will see in the following sections, smoothness is essential for an accurate *curvature analysis (CA)* of the wave-fronts and wave-backs. The IRP remains the same when V is generalized to an arbitrary scalar field.

The *isopotentials reconstruction algorithm (IRA)* is the key component of SCA, as it consumes most of the memory and time resources of the SCA. Hence, IRA has to be designed very carefully, and run on an appropriate hardware platform, to make the SCA a success. Nowdays, the CUDA-GPUs of the NVIDIA's Fermi or Tesla video cards, are an amazing such platform, which allows one to run a supercomputer on one's own laptop! We take advantage of these cards to implement a new *parallel, CUDA-based IRA (PIRA)*. To the best of our knowledge, this is the first algorithm of this kind.

PIRA belongs to a novel, CUDA-breed of algorithms, which minimize the amount of synchronization, they require to reconstruct global information, from a frame F. To achieve this goal, it divides its work in three parts: PMS) A fully parallel, local-information-computation procedure; PIE) A hierarchically parallel, global-information-computation procedure; and SOG) An optional, sequential to random-access, global-information-computation procedure.

Parallel marching squares (PMS).. The fully parallel, local-information-computation procedure PMS uses an adaptation of the *marching squares* algorithm. Given a frame F, PMS considers in parallel, that is in a different thread of a CUDA block, each 2×2 square s, of an adjacency-based $(N-1) \times (N-1)$ partition of F. Each thread, locally and accurately computes the intersection points of zero, one or two lines, with s. The number of lines and their crossing pattern with s depends on the values in the corners of s (type of s).

To determine the type of each square s of F, frame F is first subjected, in parallel, to the test $F_{i,j} \geq v$, for each index (i, j). The result of the test is stored in a boolean matrix B. The boolean value of the corners of s in B, traversed say in clockwise order, determine 16 possible intersection cases. These cases are shown in Figure 3. In summary: 1) If all corners of s in B have the same value, there is no line crossing at all. 2) Otherwise, if diagonal corners have the same value, there are two ambiguous line crossings. Ambiguity is removed by averaging the value in all corners in s and comparing it with v. 3) Otherwise precisely one line is crossing s. Our adaptation, also associates each line a direction, by requiring that the 0 corners of square s only occur to its left.

An isoline intersects s only between two corners that have opposite boolean value in B. In other words, one corner has a value less than v and the other has a value which is greater than v. The intersection points, which also represent the starting and the ending points of a directed, one-segment-long polyline, are computed via linear interpolation. For example, suppose that the corners of s at position (i, j) in B, result in bitvector 0111. Then, the line segment crossing s has the points $((x_0, y_0), (x_1, y_1))$ defined as follows:

$$x_0 = j, \quad y_0 = i + (v - F_{i,j}) \,/\, (F_{i+1,j} - F_{i,j})$$
$$y_1 = i, \quad x_1 = j + (v - F_{i,j}) \,/\, (F_{i,j+1} - F_{i,j})$$

This information is stored in the corresponding one segment-polyline at position (i,j), in an $N \times N$ matrix S of squares. The starting and ending polylines of the open polylines list are initialized to the location of this polyline. If there are

two polylines, the first one is the first in the list and the second one, the second. The list of closed polylines is initialized to null. The number of polyline segments and the number of open/closed polylines is also appropriately initialized.

```
struct Point    // typed points
{
    int    type; // 0 = row crossing, 1 column crossing
    float  x,y;  // x and y coordinates
    float  y;    // y coordinate
}

struct PolyLine                  // directed polylines
{
  int           type;            // 17 types
  int           number;          // number of segments
  Point         start, end;      // start and end points
  PolyLine *    next;            // ptr to next polyline
}
```

To speedup the interpolation process, the case analysis of the square type is efficiently pre-stored in a lookup table T, allocated in the constant memory of the GPU card.

```
struct Square                    // Data Structure for Squares
{
  int           num_of_opl;      // number of open polylines
  int           num_of_cpl;      // number of closed polylines
  PolyLine *    opl_start;       // pointer to first open polyline
  PolyLine *    opl_end;         // ptr to the last open polyline
  PolyLine *    cpl_start;       // ptr to the first closed polyline
  PolyLine *    cpl_end;         // ptr to the last closed polyline
  PolyLine      poly_lines[2]; // storage allocated on leaf level
}
```

For each square type t and for each one-segment polyline p, the table consists of the following entries: Two line coefficients and four coordinate displacements. For the above example, where $t=7$, $p=0$, table $T_{t,p}$ contains:

$$\begin{aligned} T_{t,p} = \{&\{a=0,\ b=0,\ i_1=0,\ j_1=0,\ i_0=0,\ j_0=0\},\\ &\{a=1,\ b=0,\ i_1=1,\ j_1=0,\ i_0=0,\ j_0=0\},\\ &\{a=1,\ b=0,\ i_1=0,\ j_1=1,\ i_0=0,\ j_0=0\},\\ &\{a=0\ \ b=0,\ i_1=0,\ j_1=0,\ i_0=0,\ j_0=0\}\} \end{aligned}$$

Denote $T_{t,0}$ and $T_{t,0}$ withe the corresponding field names, indexed by subscripts 0 and 1. Then the one-segment-polyline starting coordinate can be computed uniformly, as follows:

$$\begin{aligned} x_0 &= j + b_0 + a_0(v - F_{i+i_{00},j+j_{00}}) \,/\, (F_{i+i_{01},j+j_{01}} - F_{i+i_{00},j+j_{00}})\\ y_0 &= i + b_1 + a_1(v - F_{i+i_{10},j+j_{10}}) \,/\, (F_{i+i_{11},j+j_{11}} - F_{i+i_{10},j+j_{10}}) \end{aligned}$$

Parallel isoline extraction (PIE).. Using PMS one can readily plot the isolines of V. However, there is no global information available yet, despite of the fact that we know the isoline segments, their orientation, and their linking. However, we do not know: 1) What is the belonging of segments to particular isolines? 2) How many isolines are in V? 3) How many segments they contain? and 4) What is their starting and ending point. It is only our brain that connects segments in a meaningful way!

The public *contour* function of Octave uses a sequential, recursive algorithm, to obtain global information. This works as follows: 1) Traverse a matrix of interpolated segments, row by row and column by column, and pick the first unmarked one. 2) Then recursively follow matching segments in a meaningful way, until the border or the starting point is reached again (no direction is readily available as in our PMS). During this process, dump all points traversed in a dynamic array, and mark all the segments as visited. 3) Once an entire isoline is found, search for the next unmarked segment, until the last row and column is reached.

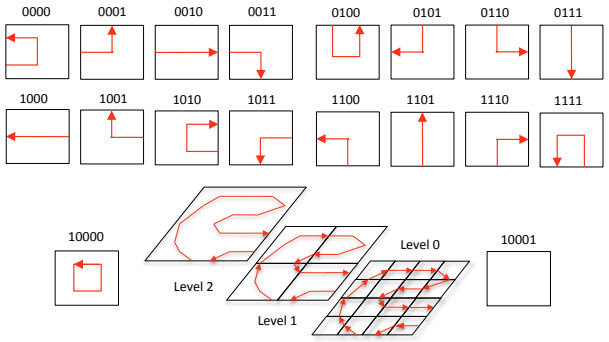


Figure 4: Isopotential extraction.

Recursion and the sequential traversal of the entire matrix of segments is however prohibitive, for a fast curvature analysis. The *isoline* function available at [12], is sequential, but not recursive. It: 1) Uses the marching squares algorithm to compute all (unoriented) segments and dump them, in no particular order, in a dynamically allocated array. 2) Then, it extracts the first unmarked segment, and repeatedly searches the entire array for an unmarked, matching continuation. Its time complexity is therefore, comparable with the one of the recursive algorithm.

To obtain a fast algorithm, we take advantage of the GPU cards. This imposes however, several important restrictions on PIE: 1) It cannot be recursive; 2) It cannot dynamically allocate memory; 3) It can only contain for-loops, with known upper bound. As shown in Figure 4, the main idea of PIE is to to organize the squares S passed by PMS, in a quad-tree fashion, such that, at every level in the tree hierarchy, sibling nodes (and their immediate children) are processed in parallel, by a different CUDA thread. Hence, global information is computed sequentially, in $log_4 N$ steps.

During this process, a parent either collects or pastes together the polylines of its children, if they have matching start and end points. A particular problem during this process is dynamic memory allocation. In the CUDA core, such allocation is not possible. Moreover, there is no way to predict in advance, without gross overestimation, the amount of memory needed in each node, to store the list of polylines it has identified so far, and the polylines themselves.

Our solution is to reuse the memory already allocated in the leaf squares, for their two one-segment-long polylines, during the upwards sweep of the quad-tree. Each time, when information is collected from the 4 children, the result is stored in child 0. This information is later on passed upwards in the hierarchy. Child 0's information gets therefore lost.

Whenever the parent process matches the ending point of polyline p of child c, with the starting point of polyline q of child d, it updates the starting and end points of polyline pq in p, and removes q from the polyline list of child d. It then continues from a copy of q to find the next match. The total number of matches, is bound by the number of polylines in all children, and this is used in a find-next-match for-loop. This loop is the equivalent of the recursive match-

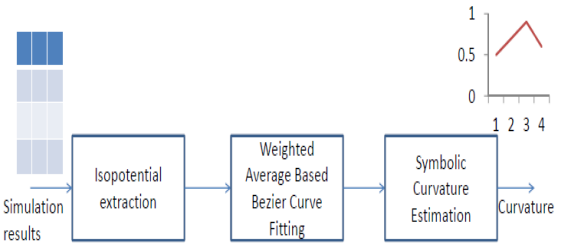


Figure 5: Block diagram for curvature estimation

next-segment search in *contour* or *isoline*, but it is limited to the *polylines* of the parent's children. This dramatically decreases the time complexity of the recursive search.

Whenever, the parent process collects the polyline lists of the children, it appends these lists to the lists of child 0. For this purpose, the Square structure has the start/end fields of the list of open/closed polylines, and each polyline has a next polyline field. The number of polylines, and the number of segments, are also updated accordingly.

In order to efficiently match the end point of a polyline p of a child c with the starting point of a polyline q, it is important to know what sabling of c might have such a q. For this purpose, we classify the polylines according to their starting and ending faces in c, as shown in Figure 4. This leads to a classification of polylines, that is similar to the one for squares. In contrast to squares however, polylines may start and end up on the same face, or they may be closed, that is, they are completely contained in c. The second case does not require further processing. Similarly, if a polyline of square c, starts and ends on a face that is not adjacent to any of the c's siblings, no processing is required at this level either. To enable this kind of analysis, a type field is added to the polyline data structure, too.

The downside of reusing the squares while collecting and propagating information, is that the linking information of one-segment-long polylines is lost. Consequently, one has to store this information in a different place. For simplicity and speedup reasons, we allocate two hash tables X and Y of size $N \times N$: the first is indexed by the integer value of horizontal intersection points, the second by the integer value of vertical intersection points. Each entry of X and Y stores the destination point and a bit classifying it as a horizontal or vertical intersection. This allows to determine whether to choose the X or the Y hashtable next. In general, unless the isolines are fractals, the number of segments is orders of magnitude smaller than $N \times N$. Hence, more information acquired about the kind of isolines to expect, may improve the size allocation, by choosing an $M << N$ for these tables.

Output generation procedure (SOG).. The optional SOG procedure selects the isolines of interest according to some given criterion, for example the longest isoline, and stores the points of these isolines in an array, sorted by their traversal order. The size of the isolines, their starting point, and their ending point are available at the root of the quadtree. This information is used to dynamically allocate the corresponding arrays, outside of the CUDA core. The X and Y hash tables are then used to traverse the associated isoline and dump the points traversed in the array. This process transforms the local, sequential-linking information, in a global, random-access information, where entry $i+1$ is known to be the successor point of entry i.

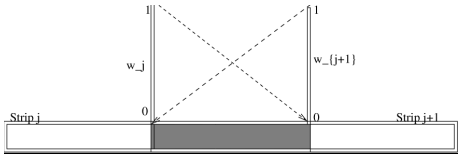


Figure 6: Weighted-average-based smoothing

We have reconstructed the isolines of 10 000 frames, both with PIRA and the *contour function* of Matlab. The first took 1.65 seconds while the second took 720 seconds. Hence, PIRA had a 444.444 fold speedup compared to *contour*! The rest of SCA has not been parallelized yet. However, this is easily accomplished as we describe in the next sections.

3. CURVATURE ESTIMATION

After obtaining the wavefront and waveback isopotentials, as produced by SOG from a sequence of frames, we proceed to estimate the curvature of these isopotentials.

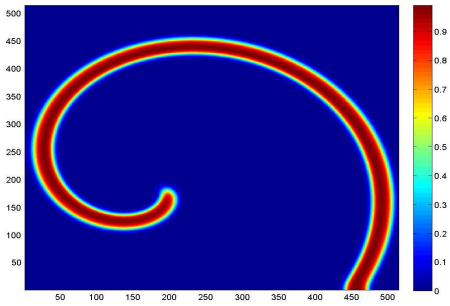
The curvature characterizes the shape of the isopotential and thus guides the detection of arrhythmias. As discussed in [8], the wave propagates at higher speed if the wavefront is concave, and at lower speed if the wavefront is convex. At particular levels of convexity, the wave stops propagating.

In order to facilitate the detection of arrhythmias, any method estimating curvature must satisfy two requirements: 1) The technique must be accurate; and 2) The method employed must provide curvature values continuously along the perimeter of the isopotential. Thus the method must be independent of the spatial resolution at which the isopotential is estimated or the grid on which the cardiac model is solved.

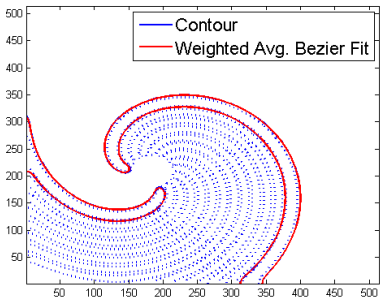
Our technique of curvature estimation can be represented by the block diagram of Figure 5. The isopotential obtained in the previous section is a series of points on $\mathbb{R}^2$. Starting from this, our method of obtaining a smooth curvature estimate involves the following steps:

1. **Preprocessing.** Divides the isopotential obtained from SOG into overlapping strips of constant length.
2. **Bézier curve fitting.** Fits each strip with a third degree Bézier curve, thus obtaining a piecewise degree-3-polynomial fit of overlapping Bézier curves.
3. **Curvature estimation.** The Curvature is computed along the Bézier curves using symbolic techniques.
4. **Overall curvature estimation.** Obtains a smooth estimate of the curvature along the entire isopotential.

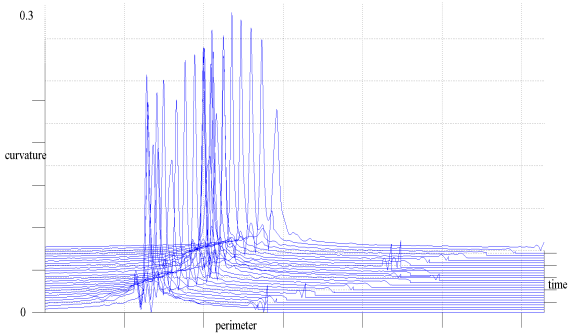
First, we divide the isopotential I_v into overlapping segments, we call *strips*. The length of each strip is controlled by the parameter STRIP_LENGTH. The percentage overlap among them is controlled by OVERLAP. This is the initial pre-processing done before polynomial fitting. The following sections describe each of the remaining steps in detail.



(a)Numerical Simulation



(b)Contour evolution



(c)Curvature Trend

Figure 7: Results for Case Study 1: Re-entry with circular core

4. PIECEWISE BÉZIER FITTING

Most of the formalisms of curvature in $\mathbb{R}^2$ involve second order derivative terms. This motivates a need to obtain at least a degree-3 polynomial approximation to the isopotential. We fit the isopotential with overlapping cubic Bézier curves in a piece-wise manner, thus satisfying the above requirements of curvature estimation.

Given a strip $I_{v,j}$, we want to obtain a Bézier curve that approximates this strip upto a certain degree of L2 error. A Bézier curve approximation for $I_{v,j}$ will have the form below, where parameter t varies in the range $[0, 1]$:

$$X_j(t) = (1-t)^3 P_j^0 + 3t(1-t)^2 P_j^1 + 3t^2(1-t)P_j^2 + t^3 P_j^3 \quad (1)$$

$$Y_j(t) = (1-t)^3 Q_j^0 + 3t(1-t)^2 Q_j^1 + 3t^2(1-t)Q_j^2 + t^3 Q_j^3 \quad (2)$$

Here, P_j^0–P_j^3 and Q_j^0–Q_j^3 are the control points of the Bézier curves that approximate the curve along x and y axis, respectively. To sample from this curve, we evaluate the above expressions over the interval $t \in [0, 1]$.

P_j^0, P_j^3, Q_j^0 and Q_j^3 are the terminal control points which coincide with the data. The fitting procedure adapted from [9] and [15] optimizes the intermediate control points P_j^1, P_j^2, Q_j^1 and Q_j^2 to minimize the least squares error. We assume a uniform parametrization of t in $[0, 1]$ for each segment. Thus, the error functions, while fitting $I_{v,j}$ are:

$$E_x = \sum_{i=1}^{SL} [x_i - X_j(t_i)]^2, \quad E_y = \sum_{i=1}^{SL} [y_i - Y_j(t_i)]^2$$

Replacing equations (1) and (2) in (3) and (4), respectively, we obtain the following set of equations:

$$E_x = \sum_{i=1}^{SL} [x_i - (1-t_i)^3 P_j^0 + 3t_(1-t_i)^2 P_j^1 + 3t_i^2(1-t_i)P_j^2 + t_i^3 P_j^3]^2$$

$$E_y = \sum_{i=1}^{SL} [y_i - (1-t_i)^3 Q_j^0 + 3t_i(1-t_i)^2 Q_j^1 + 3t_i^2(1-t_i)Q_j^2 + t_i^3 Q_j^3]^2$$

The following calculations are shown only for E_x. For E_y the expressions follow from those for E_x. P_j^1 and P_j^2 can be obtained at the minimum value of E_x by usin:

$$\frac{\partial E_x}{\partial P_j^1} = 0, \quad \frac{\partial E_x}{\partial P_j^2} = 0$$

Solving the above two equations we obtain the following expressions for P_j^1 and P_j^2:

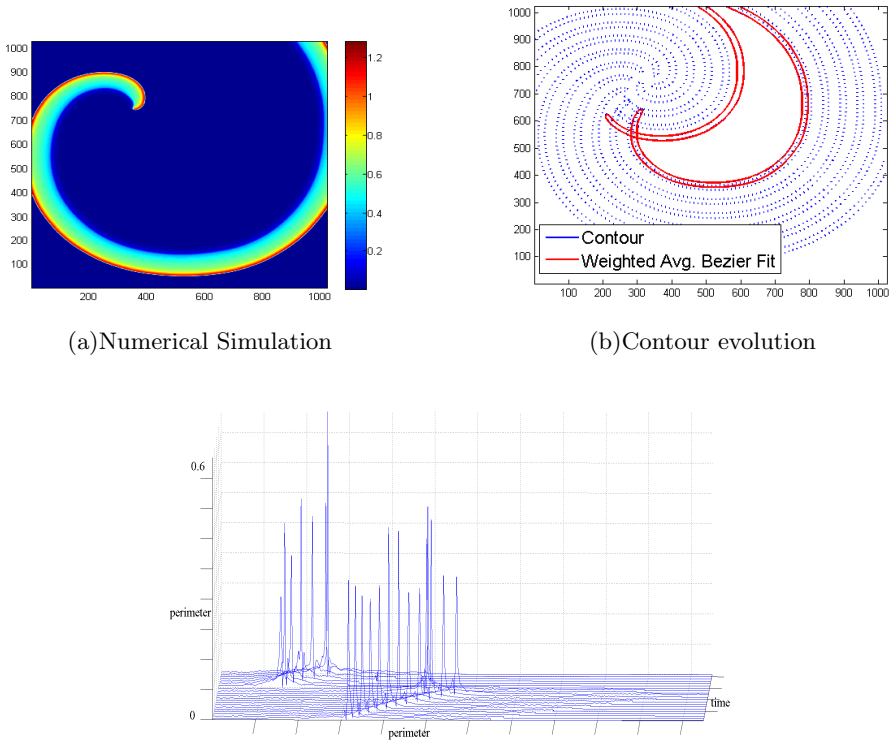
$$P_j^1 = \frac{\alpha_2^j \beta_1^j - \alpha_3^j \beta_2^j}{\alpha_1^j \alpha_2^j - {\alpha_3^j}^2}, \quad P_j^2 = \frac{\alpha_1^j \beta_2^j - \alpha_3^j \beta_1^j}{\alpha_1^j \alpha_2^j - \alpha_3^{2j}}$$

where $\alpha_1, \alpha_2, \alpha_3, \beta_1$ and β_2 for each segment are given by:

$$\alpha_1 = 9 \sum_{i=1}^{SL} [t_i^2 (1-t_i)^4]$$

$$\alpha_2 = 9 \sum_{i=1}^{SL} [t_i^4 (1-t_i)^2], \quad \alpha_3 = 9 \sum_{i=1}^{SL} [t_i^3 (1-t_i)^3]$$

$$\beta_1 = 3 \sum_{i=1}^{SL} [t_i (x_i - (1-t_i)^3 P_0 - t_i^3 P_3)(1-t_i)^2]$$



(a)Numerical Simulation

(b)Contour evolution

(c)Curvature Trend

Figure 8: Results for Case Study 2: Re-entry with hypocycloidal core

$$\beta_2 = 3\sum_{i=1}^{SL}[t_i^2(x_i - (1-t_i)^3 P_0 - t_i^3 P_3)(1-t_i)]$$

The above procedure explains how cubic Bézier curves are fit on one strip of the isopotential. To make the curvature estimates smooth, we fit the curves on overlapping segments.

5. WEIGHTED-AVERAGE IMPROVEMENT

The fitting procedure described above is seqential in nature. However, for every strip selected, one can run this procedure in parallel, by using a different CUDA-thread on the CPU-cards. Although we did not do this yet, this is a simple adaptation of this algorithm.

After each strip is fit the data with cubic Bézier curves as explained above, we improve the smoothness of the overall fit. A pure piece-wise fitting approach would create discontinuities in the derivative of the isopotentials at the points where the curves meet. We ensure that derivatives upto second order are well defined everywhere on the iso-line, by smoothening the overlapping strips. Parameter OVERLAP determines the percentage overlap between adjacent strips.

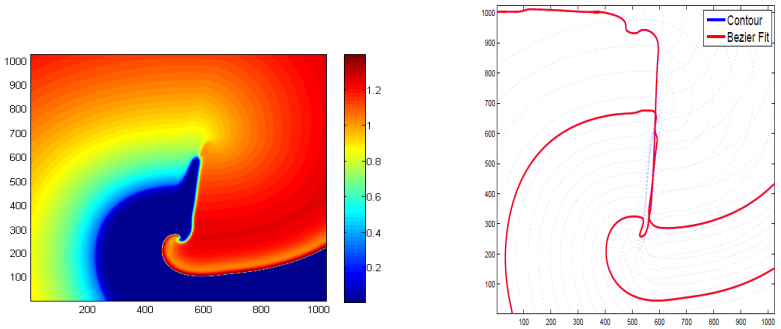
Consider a part of the isopotential that has two adjacent overlapping strips with indices j and j+1. To define the Bézier curve fit for this part of the iso-line, we use a weighted average based method. Suppose the curves describing the two strips are f_j and f_{j+1}, then the fit for the two strips is given by $w_j f_j(t_j) + w_{j+1} f_{j+1}(t_{j+1})$, where $w_{j+1} + w_j = 1$. In essence, the influence of the adjacent curves on the fit is gradually varied in the region of overlap. In the region where there is no overlap, the fit is completely described by the only Bézier curve corresponding to that strip. As one enters the region of overlap, the fit is a weighted average of the two Bézier curves corresponding to adjacent strips. The weights are varied linearly in our scheme as shown in Figure 6. The same weighted average smoothing is performed for the derivatives and curvature values.

It should be noted that this smoothing results in a fit that is $C2$ continuous. Each Bézier curve is a third degree polynomial. In the overlapping region, the fit is a weighted average of two cubic polynomial functions. Thus even in the overlapping region the fit is C^2 smooth. Using this weighted average based fitting on overlapping strips, we proceed to estimate curvature in the next section. Smoothening can also be performed in parallel for every overlap.

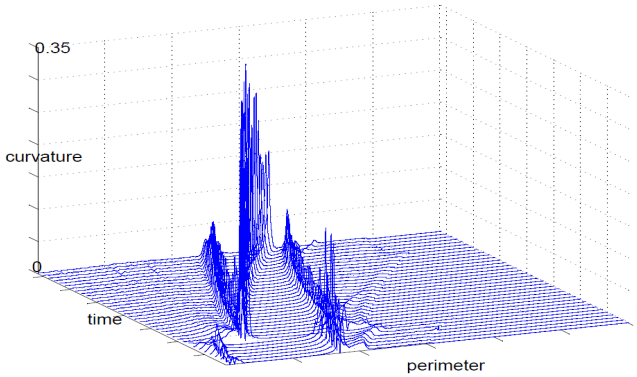
6. SYMBOLIC CURVATURE ESTIMATION

We use symbolic computations in MATLAB to evaluate the curvature along the isopotential. This constitutes step 2 of our method listed above. The fitting procedure described above provides smooth Bézier functions that describe the isopotentials. As these are closed form expressions, they can be manipulated symbolically. MATLAB's symbolic toolbox provides the facility to declare symbolic variables, construct functions out of them and operate on those functions. Once the operations yield the expressions of interest, they can be evaluated at arbitrary resolution by suitably specifying the interval for the symbolic variables.

In our case, we obtain the functions $X_j(t)$ and $Y_j(t)$ for each strip. The curvature of this strip of the isopotential is derived using elementary calculus:



(d)Numerical Simulation (e)Contour evolution



(f)Curvature Trend

Figure 9: Results for Case Study 3: Re-entry with linear core

$$\kappa_j(t) = \frac{|r'_j(t) \times r''_j(t)|}{|r'_j(t)|^3} \tag{3}$$

where $r_j(t) = [X_j(t), Y_j(t)]$ is the position vector described by the Bézier curve. The important point to note is that $X_j(t)$, $Y_j(t)$ and thus $r_j(t)$ are managed as symbolic expressions which are functions of the symbolic variable t. Thus $\kappa_j(t)$ is obtained in closed form as an expression in t. Symbolic operations on $r_j(t)$ is performed using MATLAB's symbolic math toolbox. [11]

After obtaining closed form expressions for curvature, their continuity ensures that we can evaluate them at any resolution of the parameter t. This translates to obtaining continuous estimate of curvature along the perimeter of the isopotential. Note that one can precompute the symbolic form of coefficients, which can be thereafter instantiated according to the sampled data. Hence the symbolic Matlab tool-box is not indispensable.

Step 3 of our method evaluates these curvature functions along the isopotential using the weighted approach technique described in the previous subsection. Currently we maintain uniform resolution for t along all the strips. Adapting this to the shape of the iso-line, is part of our future work. In particular, the information stored in the quad-tree of PIE, for example the filling factor of the area associated to its nodes. might facilitate a fast and accurate breakup of the isoline in isoline-strips, improving on the idea in [8].

The runtime of the curve-fitting and curvature-calculation routines depends on the length of the isopotential. The extraction process checks each grid square for a possible point of the isopotential. The number of edges on a $n \times n$ grid can be calculated by solving the following recurrence:

$$E(n) = E(n-2) + 4n + 4(n-3) + 8 \quad E(1) = 4, \quad E(2) = 8 \tag{4}$$

The solution is given by

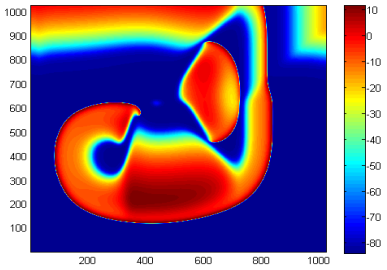
$$E(n) = -2(-1)^n + 2n(n+1) - 2 \tag{5}$$

Hence the maximum length of the iso-line is $O(n^2)$, which bounds the number of operations in the fitting and curvature routines to $O(n^2)$

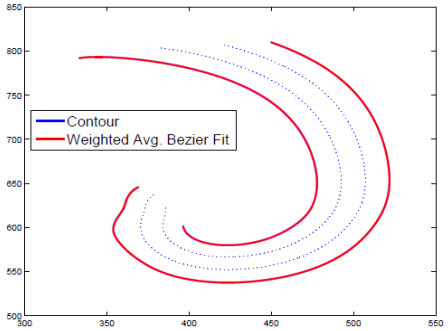
7. CASE STUDIES

We apply the above methods of isopotential reconstruction and curvature estimation on four case studies that simulate different forms of arrhythmia. For each case, we extract the isopotential in every frame and fit it using the weighted average based Bézier curve method. In the following figures, we show both the fit and the original isopotential only for the first and last step of evolution. Curvature trends along time that characterize the different arrhythmia are calculated. We discuss our results in the following subsections.

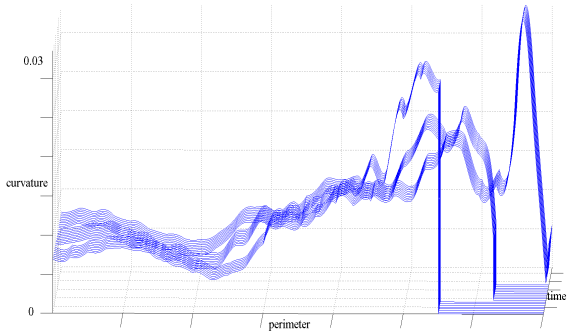
1. Re-entry with circular core.. Re-entrant spiral waves in atrial chambers are precursors for ischemic stroke and fibrillation. In this case study we simulated the Barkley model on a tissue of 514×514 cells. Time scale used was 10ms.



(g)Numerical Simulation



(h)Contour evolution



(i)Curvature Trend

Figure 10: Results for Case Study 4: Spiral Break-up

The core of the spiral-shaped-excitation waves traces a circular trajectory. After initial excitation, the spiral-shaped isopotentials were extracted at a scaled level of 0.7.

The results are shown in the Figure 7. Subfigure (b) shows the fit and the original iso-line for the first and last time step. As the spiral rotates, its tip traces a circular trajectory, the basic shape of the spiral isopotential does not change. Thus we obtain a curvature trend that remains almost same along time. The region of maximum curvature corresponds to the spiral tip which remains around the center of the iso-potential throughout the simulation. The rest of the isopotential shows a relatively lower curvature.

2:. Re-entry with Hypocycloidal core. As we see in Figure 8, the curvature of the iso-potential is similar to that in case study 1. The difference is due to the motion of the spiral. In this case study, a tissue of size 1024×1024 was simulated using the MHA model. The tip of the spiral traces a hypocycloidal trajectory. The isopotential was extracted again for 0.7 and the time scale of the simulation was 10ms.

As the tip rotates, the length of the wavefront changes and at times, the tip is not the center of the isopotential. The turn of the spiral is evident in the curvature trend of Figure 8(c). As the length of the spiral changes, the region of highest curvature shifts on the curvature trend. Thus, the trend of morphological features like curvature, can capture the dynamics of cardiac arrhythmia.

3. Re-entry with Linear Core. This case study further exemplifies and reinforces the idea of capturing tachycaridia dynamics using curvature. For this case study, a tissue of 1024×1024 cells was simulated under the minimal model at a time scale of 10ms. Isopotential extraction was done for a scaled level of 1.0. This study analyses curvature of cardiac excitation in the case of an obstacle. Any resistance in the path of electrical propagation can result in the spiral tip to start following a linear trajectory. The results obtained are shown in Figure 9.

The basic shape of the spiral in this case contains three regions of high curvature. The highest curvature is found at the separation of the wavefront and waveback. As the tip moves along the linear path, the separation between curvature peaks, corresponding to the high curvature regions, changes. The highest peak starts near to the left peakm that corresponds to the first high curvature bend of the isopotential. With time, as the tip moves down to the other end of the linear path, the central curvature peak shifts towards the right peak.

4. Spiral Wave Breakup. In this case we study the onset of fibrillation. The spatio-temporal definition of myocardium fibrillation involves the break-up of re-entrant waves. This breakup creates daughter spirals which interact to produce emergent behaviour. Thus predicting the spiral break-up is crucial to the problem of predicting fibrillation. A tissue of 1024×1024 cells was simulated using the Beeler-Reuter model. Spiral break-ups occur at a very short time scale.

This the model was simulated at 1ms time scale to allow for detection of abrupt changes in isopotential morphology.

Subigure 10(a) shows one simulation frame where the first breakup has already occurred. We track the isopotential of value 4.0 till the first breakup occurs. As we approach the moment of detachment, the isopotential shows a dent near the site of break-up. Thus change in shape translates to creation of a high curvature region. The changing shape of isopotential is shown in Subigure 10(a). Again, the isopotential and the polynomial fit is shown for the first and last step and intermediate steps are shown in dashed lines.

Subigure 10(c) shows the trend of the curvature as the isopotential evolves towards breakup. Just before detachment, we see high curvatures corresponding to the evolving site of break-up. This shows that tracking curvature can provide clues for predicting onset of possibly fatal fibrillation.

8. CONCLUSIONS

The technological developments within the graphical-processors community, NVIDIA in particular, coupled with the theoretical advances in the computer-aided verification community, have set the stage for fast simulation, powerful analysis and accurate prediction of complex biological processes. In this paper we have present the basic components of such a combination: 1) Model checking-based parameter-space partitioning, for efficient and principled parameter selection; 2) Parallel simulation algorithms; and 3) Parallel curvature-analysis algorithms, of the frames generated thorough simulation or optical mapping. Our algorithms take advantage of the NVIDIA's graphical cards Tesla and Fermi, and the associated CUDA architecture. Our results show a 444.444 speedup of isopotential reconstruction compared to Matlab-based contour algorithm. Our case studies identified promising signatures, of various forms of cardiac disorders, which may be used to predict their onset. We are currently working to parallelize the rest of our spiral-waves-classification algorithm (SCA) and to expand the variety of our case studies.

9. REFERENCES

[1] A. Bueno-Orovio, M. E. Cherry, and F. H. Fenton. Minimal model for human ventricular action potentials in tissue. *J. of Theor. Biology*, 253(3):544–560, 2008.

[2] E. M. Cherry and F. H. Fenton. A tale of two dogs: Analyzing two models of canine ventricular electrophysiology. *American Journal of Physiology - Heart and Circulatory Physiology*, 292:H43–H55, 2007.

[3] E. M. Cherry and F. H. Fenton. Visualization of spiral and scroll waves in simulated and experimental cardiac tissue. *New Journal of Physics*, 10:125016, 2008.

[4] F. H. Fenton and E. M. Cherry. Models of cardiac cell. *Scholarpedia*, 3:1868, 2008.

[5] R. Grosu, G. Batt, F. Fenton, J. Glimm, S. S. C. Le Guernic, and E. Bartocci. From cardiac cells to genetic regulatory networks. In *Proc. of CAV'11, the 23rd International Conference on Computer Aided Verification*, LNCS, Cliff Lodge, Snowbird, Utah, USA, July 2011. Springer.

[6] V. Iyer, R. Mazhari, and R. L. Winslow. A computational model of the human leftventricular epicardial myocytes. *Biophysical Journal*, 87(3):1507–1525, 2004.

[7] D. L. Jones and R. J. A. et al. Heart disease and stroke statistics 2010 update: A report from the american heart association. *Circulation*, December 2009.

[8] M. Kay and R. Gray. Measuring curvature and velocity vector fields for waves of cardiac excitation in 2-d media. *IEEE Transactions on Biomedical Engineering*, 52(1):50–63, January 2005.

[9] D. M. Khan. Cubic bezier least square fitting. *Matlab File Exchange*, July 2009.

[10] C. H. Luo and Y. Rudy. A dynamic model of the cardiac ventricular action potential. I. Simulations of ionic currents and concentration changes. *Circulation Research*, 74(6):1071–1096, June 1994.

[11] Mathworks. Matlab symbolic math toolbox, 1986. http://www.mathworks.com/products/symbolic/.

[12] Matlab. Central's isocontour algorithm., 1986. http://www.mathworks.com/matlabcentral/fileexchange/30525-isocontour.

[13] C. J. Myers. *Engineering Genetic Circuits*. CRC Press, 2010.

[14] L. Priebe and D. J. Beuckelmann. Simulation study of cellular electric properties in heart failure. *Circulation Research*, 82:1206–1223, 1998.

[15] D. F. Rogers and J. A. Adams. *Mathematical Elements of Computer Graphics*. McGraw-Hill Science/Engineering/Math, New York, 1989.

[16] K. H. Ten Tusscher, D. Noble, P. J. Noble, and A. V. Panfilov. A model for human ventricular tissue. *American Journal of Physiology*, 286:H1573–H1589, 2004.