Implications of the Turing Completeness of Reaction-Diffusion Models, informed by GPGPU simulations on an XBox 360: Cardiac Arrhythmias, Re-entry and the Halting Problem

S. Scarle[1,*]

[1]*Rare Ltd., Manor Park, Twycross, Warwickshire, CV9 3QN, UK.*

(Dated: November 25, 2008)

Abstract

In the arsenal of tools that a computational modeller can bring to bare on the study of cardiac arrhythmias, the most widely used and arguably the most successful is that of an excitable medium, a special case of a reaction-diffusion model. These are used to simulate the internal chemical reactions of a cardiac cell and the diffusion of their membrane voltages. Via a number of different methodologies it has previously been shown that reaction-diffusion systems are at multiple levels Turing complete. That is, they are capable of computation in the same manner as a universal Turing machine. However, all such computational systems are subject to a limitation know as the Halting problem. By constructing a universal logic gate using a cardiac cell model, we highlight how the Halting problem therefore could limit what it is possible to predict about cardiac tissue, arrhythmias and re-entry.

All Simulations for this work were carried out on the GPU of an XBox 360 development console, and we also highlight the great gains in computational power and efficiency produced by such general purpose processing on a GPU for cardiac simulations.

Keywords: heart, re-entry, cardiac arrhythmias, excitable media, halting problem, GPGPU

[*]Electronic address: `sscarle@rare.co.ukTelephone:+44(0)1827883400`

I. INTRODUCTION

Propagating waves of electrical excitation are the fastest form of long range internal communication available to animals, and so are one of the most practical applications of a subset of reaction-diffusion systems, the excitable medium. In the heart these electrical waves (action potentials) initiate contraction of the cardiac muscle. Their abnormal propagation can lead to cardiac arrhythmias, with the most dangerous of these being ventricular tachycardia (VT) and ventricular fibrillation (VF) [1]. These arrhythmias can result in sudden cardiac death, which is the largest categorical cause of death in the industrialized world [1–3].

On a number of levels reaction-diffusion (RD) systems have been shown to be capable of computation, and thus to be Turing complete. However, with such completeness comes the Halting problem and in this work we point out that this will prevent us from predicting complete re-entry in an arbitrary cardiac tissue as this amounts to solving a reversed form of the Halting problem.

In a nice link Turing himself published work on the use of RD systems in a biological context, to model pattern formation on the skins of animals [4].

A further point of interest is that all simulations for this work were carried out on the Graphical Processing Unit (GPU) of an XBox 360 development console, a variant of the popular games console used by games developers. We further highlight how such general processing on the GPU (GPGPU) techniques can bring huge increases in computational efficiency, and that this is part of an on going trend of such *serious* uses of graphical and gaming hardware [5].

II. CARDIAC MODELLING

An excitable medium is a non-linear system which has the capacity to propagate a wave of some description, and which cannot support the passage of another wave until a certain amount of time has passed (the refractory period). Cardiac tissue can be modelled as an electrically excitable medium which supports travelling waves of electrical activation, so called action potentials (APs). Such propagating APs can be described by a non-linear reaction diffusion equation [6] leading to the mono-domain tissue model

$$C_m \frac{\partial V_m}{\partial t} = \nabla.\mathbf{D}\nabla V_m - I_{ion} \tag{1}$$

The left hand side of Equation 1 gives the current due to the capacitance of the cell membrane, whilst the right gives current due to both gradients in trans-membrane potential (diffusive term = $\nabla.\mathbf{D}\nabla V_m$) and ion channels, pumps and transporters in the cell membrane (reaction term = I_{ion}). Where, $\mathbf{D}$ is the diffusion tensor, V_m is voltage across the cell membrane, C_m is membrane capacitance per unit membrane area and I_{ion} is membrane current flow per unit area.

RD systems are so called as they can be seen to represent reactive chemical species which are not well mixed. This means that to fully explain their behaviour we must take into account both the diffusion of the species through space as well as their reactions with each other. An excitable medium used to represent cardiac tissue can be seen as a *simple* type of reaction-diffusion system, in that usually only one *chemical species*, the membrane voltage, is allowed to diffuse.

III. COMPUTATION AND THE HALTING PROBLEM

Alan Turing is commonly accepted as the creator of modern computer science with amongst other things his landmark paper on the *Entscheidungsproblem* [7]. In this paper he created the concept of what is now called a Turing machine. A hypothetical device which would be capable of carrying out any conceivable mathematical problem if it could be represented as an algorithm. This in turn lead to the idea of the Turing-Completeness of a system if it were capable of doing all such computation. Critically, Turing also highlighted the Halting problem in this work. This states that it is impossible to construct a general algorithm which will state whether an arbitrary program will terminate or continue forever when run on a Turing-Complete device/system.

IV. LEVELS OF COMPUTATION WITHIN REACTION-DIFFUSION SYSTEMS

Reaction-diffusion or excitable medium computational devices represent data in the distribution of their chemical species. The actual computation is implemented by excitation waves, travelling and interacting in the medium, and such wave-based information processing in excitable chemical media was first achieved with pioneering work on image processing using a light-sensitive Belousov-Zhabotinsky reaction [8, 9].

In this section we outline a number of studies in the literature which have carried out computation with or have shown computational traits in RD systems. These can be divided into three types; problem solving using an RD directly, embedding of logic gate structures into an RD system and finally more theoretical work showing that any RD-like systems are capable of computation. This last type leading to the Reaction Diffusion Machine (RDM).

A. RD as problem solver

It is now well known that RD chemical systems have a unique ability to efficiently solve combinatorial problems with their innate parallelism [10]. Since both the data and the results are encoded as concentration profiles of reagents, with the computation being preformed via the spread and interaction of wave-fronts (see overview [11–13]). This was first experimentally implemented to carry out basic image processing using the light sensitive Belousov-Zhabotinsky (B-Z) reaction [8, 9, 14]. Several other experimental and simulated prototypes of excitable media computing devices or reaction-diffusion processors have been designed and tested on a variety of tasks, including; path planning [15–18], robot navigation [19, 20]. Further, work on slime moulds navigating labyrinths [21, 22] has suggested that methods akin to these are used in nature. This work having the *honour* of being awarded the Ig-Noble prize for Cognitive Science in 2008.

B. RD as logic gate system

The fundamental building blocks of a computer are logic gates, these equate at a basic level to the logic statements of AND, OR and NOT, *i.e.* an OR gate outputs TRUE if input 1 **or** input 2 are TRUE else it outputs FALSE. Table I shows inputs and the produced output for the most common gates. In a modern digital computer, TRUE = 1 and FALSE = 0, and from this binary arithmetic is possible.

Two logic gates are of particular importance: Not AND (NAND) and Not OR (NOR). These are referred to as the universal logic gates as the function of any logic gate can be constructed with purely NAND or NOR gates (see Figure 1), and therefore by extension any logic circuit.

In a more direct approach such logic gate circuits have been implemented via RD systems in laboratory experiments [23], as well as in numerical simulations [10, 23–27]. Further computational systems of higher level function have been constructed/modelled: being capable of counting [28] and an implementation of a simple memory unit [27, 29].

C. Reaction Diffusion Machine

A computational model inspired by RD phenomena, called the Reaction-Diffusion Machine (RDM) was introduced by Simone and Bandini [30], and then expanded to produce the Multi-layered Reaction-Diffusion Machine (MRDM) [31]. An RDM allows for the simulation of complete systems in which entities react locally with each other and with the environment, and the global system behaviour emerges from the local behaviour of the component entities. In an RDM control is fully distributed, as behaviour is determined by local computation based on position and sensitivity to fields as well as on reaction and diffusion patterns. The MRDM, within appropriate constraints, collapses to a standard Cellular Automaton (CA).

Later work [32] provided the first theoretical assessment of an RDM, through a comparison with the standard Turing machine model. They constructed a theoretical RD system capable of simulating a Turing machine, what they called a Turing RDM (TRDM). However, their TRDM requires three fields to freely diffuse through its network of compartments. Cardiac models, on the whole, have only one such field, excitation / membrane voltage, meaning that it is unlikely that a cardiac model could directly be used as a TRDM.

However, it may still be possible to carry out computation directly in a cardiac model by constructing logic gate systems, and in the rest of this work we shall show how we went about constructing such systems using the Fenton-Karma four variable cardiac cell model. If it can be shown that a cardiac model is capable in some way of carrying out computation in a Turing-Complete manner, then implications of the Halting problem may be felt and by inference it will also impinge on to the behaviour of cardiac tissue, or at least what it is possible to predict about such tissue.

V. GPGPU

In recent years the introduction of programmable GPUs and therefore shaders to run on them has led to a rapid increase in the fidelity of real-time graphical applications, particularly in the field of games. Although non-photo realistic rendering predates sophisticated GPUs some game designers have deliberately used a non-realistic style to differentiate themselves from the realistic crowd. For example the inked graphical novel look of Crackdown [33], and the painted styles of Okami [34] and Prince of Persia [35]. However, even these styles are carried out via complex shader calculations [36, 37]. driven by demands for increased visual fidelity, GPUs have evolved into increasingly general purpose computing devices, with a growing trend to leverage this power for non-graphical applications. Leading to the technique of General-Purpose computation on the GPU or General-Programming on the GPU (GPGPU) becoming a growing trend in the field of high-level computation [38, 39].

Modern GPUs are highly optimized parallel computing devices, but they are obviously highly optimized for the calculations required for graphics *i.e.* massively SIMD (Single Instruction Multiple Data) and emphasis on vector arithmetic operations. However, if one can restate your problem in a form usable by the GPU you can still exploit this power. In fact Graphical Technology firm nVIDIA have recently released a framework for carrying out such general calculations on its hardware, with its Compute Unified Device Architecture (CUDA) [40, 41]. Other players in graphical technology (ATI, OpenGl, DirectX 11) have also constructed their own replies to this, along with chip manufacturer Intel with their GPU-CPU hybrid Larrabee.

Gaming hardware has been at the forefront of the applications of GPGPU methods and has increasingly been used for high power computing. One of the earliest examples being the creation of a supercomputer at the National Center for Supercomputing Applications at the University of Illinois at Urbana-Champaign from 70 PlayStation 2 consoles. A more up to date project used 16 networked PlayStation 3 (PS3) consoles to calculate gravity waves from black hole collisions, using the consoles Cell chip, not a GPU but architecturally similar. Meanwhile, other researchers used high-end graphics cards usually found in games PCs to simulate the repulsion between two electrons in an atom. Both of these being highly non-trivial calculations in their respective fields [42]. Meanwhile the Barcelona Supercomputing Centre, one of the most powerful supercomputing resources in the world, have investigated the possibility of using GPUs in future upgrades [43].

This may surprise the causal observer, who may think that the average game console is basically an over priced toy. However, almost by definition the current generation of games consoles and PCs are the most powerful *bangs per buck* computing hardware one can purchase. The cell chip at the heart of the PS3 is the CPU now used in IBM's high-end systems for example, whilst the XBox 360 contains a respectable Xenon - triple core 3.2 GHz PowerPC processor and a Custom ATI 500 MHz GPU.

An estimate of the potential computational gains of using GPGPU techniques over running on the CPU can be found by roughly calculating the maximum number of floating point operations the CPU and the GPU of the XBox can theoretically carry out in a second. Here measured in Gflops (10^9 floating point operations) per second. The XBox CPU has three cores each capable of one vector and one scalar operation per clock tick, *i.e.* $3 \times (5+1) \times 2 = 36$ flops. With a clock frequency of 3.2 GHz this gives us 115.2 Gflops per second. Meanwhile, the XBox GPU has three execution or arithmetic and logic units (ALUs) capable of 16 vector and 16 scalar operations per clock tick, *i.e.* $3 \times 16 \times (4+1) \times 2 = 480$ flops. With a clock frequency of 500 MHz this gives us 240 Gflops per second. Therefore in principle the GPU is capable of just over twice as many flops per second as the CPU. However, both of these maximums are in practice impossible to obtain, with the CPU unlikely to achieve 50%, although due to its more optimized nature the GPU can get far closer.

VI. LOGIC GATE SYSTEMS IN CARDIAC MODELS

In the simulations carried out for this work we have used the Fenton-Karma 4 variable model (FK4v) for I_{ion}[44].

$$I_{ion} = \frac{J_{fi} + J_{so} + J_{si}}{C_m} \tag{2}$$

with each J_x being a simulated flow of ions across a cardiac cell membrane. These are described in Table II. Fenton and Karma choose to use this naming scheme so as to reiterate that the currents do not actually represent measured currents, but only their activation, inactivation and reactivation dynamics. It is these dynamics which are needed to qualitatively reproduce cardiac cell restitution properties. The J_x are governed by three internal variables v, w, and d, and V_m rescaled to be a dimensionless and normalized activation, U.

$$U = \frac{V_m - V_0}{V_{fi} - V_0} \tag{3}$$

where $V_0 = -85$ mV (the resting potential), $V_{fi} = 15$ mV (the Nernst potential of the fast interval current). Further, $C_m = 1\ \mu\text{Fcm}^{-2}$ and $D = 0.0005\ \text{cm}^2\text{ms}^{-1}$.

All simulations for this study were carried out by solving Equation 1 in terms of U, and the ODEs for v, w and d using a Euler finite difference numeric integration with a two speed time-step. The spatial and temporal steps used were $\delta x = 0.02$ cm and $\delta t = 0.1$ ms or 0.01 ms. The choice of time-step for the integration of the variables being dependent on the rate of diffusion of the voltage at the point under consideration; if $DV_m(diff) + DV_m(int) < 0.001$ then we use ten of the shorter time-steps as opposed to one of the longer ones. With $DV_m(diff)$ being the change in voltage due to the diffusion component, and $DV_m(int)$ being the change in voltage due to the internal ionic component.

In general most abnormal behaviour of cardiac tissue is brought about by damage of, or disease in, cardiac cells. This reduces their excitability and/or their connectivity to their neighbours and hence their diffusion coefficient. Therefore if we can show that we can produce a universal logic gate using modelled cells with either of these defects, then it should be possible with real diseased or damaged cells. This in turn means that any logic circuit can thus be constructed, and that cardiac tissue is in a sense Turing complete.

A. Conversion to the XBox 360 & on to GPGPU

In our previous work [45] a highly customizable cardiac dynamics code was constructed in C^{++}. For this current work we reimplemented this code on an XBox 360 development kit. Firstly as a standard CPU based programme, and then further as a GPGPU programme. Namely, a set of vertex and pixel shaders in High Level Shader Language (HLSL), with a C^{++} driver programme

At a basic level, a standard graphical programme for a GPU comprises of vertex and pixel shader programmes or shaders for short. The vertex shader transforms an object's vertex coordinates from its own space to that of viewpoint given by the "camera". This information is then passed on to the pixel shader, along with further information related to each vertex, *e.g.* a surface coordinate system given by normal, tangent and bi-tangent, or texture (UV) coordinates. A texture being an array of colour data, essentially an image. The pixel shader then deals with rasterisation. The data from three vertices of a triangle are extrapolated across its surface and then used to calculate the required colour of the final pixel.

As much as a modern technique such as GPGPU can be said to have a *tradition*, our

GPGPU implementation of the FK4v model differs from the traditional GPGPU methodology. In a standard GPGPU calculation [38] a set of vertices are passed to the GPU representing a quadrilateral oriented parallel to the image plane so as to cover a rectangle of pixels the size of the required output array. The rasterization process produces a computational fragment for each pixel location, an instance of the pixel shader, these can read from arbitrary memory locations (with texture reads) but can only write out to memory locations defined by the pixel in the frame buffer. This output can be the final output of the calculation or it can be stored in a texture and used in further computation. Complex algorithms may require several passes of different pixel shaders to produce a final output.

We instead extensively exploit the additional functionality of the XBox GPU: flexible fetching and memory export. The XBox 360 gives full control over the fetching logic of model vertices from the vertex buffer. The data storage structure of the GPU for such vertices. Rather than specify a fixed vertex declaration up front, you can arbitrarily fetch from any location in the vertex buffer. This is based on a index from an index buffer, from which can be determined what data is required from the buffer. While memory export is essentially improved memory access and can be see as the reverse of flexible fetching allowing data to passed back into the vertex buffer at any desired location.

We therefore fully store our cardiac system on the vertex buffer. Our simulation vertex structure is comprised of four sets of four data items and is defined in HLSL as follows: **struct vertex{ float4 Position : POSITION; float4 FK4v : NORMAL; float4 Neigh : TEXCOORD0; float4 DVm : TEXCOORD1;};**

Position contains the Cartesian coordinates of the cell (x, y, z) and its membrane voltage V_m. FK4v contains the further FK4v variables (v, w, d) and its cell type, the FK4v having parameter sets for endocardial, mid-endocardial and epicardial cells. Neigh contains the indices in the vertex buffer of the four neighbours of this cell. Finally, DVm is used as a calculation space for diffusion as it contains $DV_m(int)$ and $DV_m(diff)$ with the two remaining items being held spare for the possible expansion of this code to deal with 3-D simulations were they would contain the indices of the two remaining neighbours.

GPUs are good for embarrassingly parallel problems *e.g.* rendering 3D graphics. In so far as the final colour of a given pixel is not per-se dependent on the final colours of its neighbours. However, due to the diffusion of the membrane voltage a cardiac tissue simulation is not quite embarrassingly parallel. Hence to perform the required synchronization, we broke up our calculation into a set of vertex shaders, and passed the vertex buffer through each in turn. These three shaders were called; Zapp, Diffusion and Simulation.

Zapp was closest to traditional GPGPU applications in so far as it used a texture. The C^{++} wrapper read in from the same basic input file used by our original code on when/how to apply next the stimulation to simulation. This data was written out to a linear texture's RGB components (alpha being unused). Red contained the index of the cell to be stimulated, green the type of stimulation (see Table III) and blue contained a value used by the stimulation as also described in Table III. An index buffer counting up to the number of cells excited is passed to the vertex shader, for each index the pixel of the same value was read in and then the appropriate cell's vertex Position and FK4v are read in, the stimulus applied and then the vertex was updated via memory export.

Diffusion used flexible fetching to obtain the position values of each of a cell's neighbours using the neigh component of the vertex, totaled up their V_m values and this was memory exported into the $DV_m(diff)$ component of the cell vertex. This summation is required as part of the standard Euler integration of the diffusion component of Equation 1:

$$\frac{\partial^2 V_m}{\partial x^2} \approx \frac{\sum V_m^i - 4V_m}{\delta x^2} = \frac{DV_m(diff)}{\delta t} \tag{4}$$

The final shader, Simulation, used DV_m to calculate and apply the diffusion term and then carried out updates to v, w, d and V_m due to ion currents as per Equation 1, using the same variable time step as mentioned previous. $DV_m(int)$ and all other values of the vertex were then updated using memory export.

B. Construction of a NOR gate in FK4v

Of the two universal logic gates we choose to construct a NOR gate within a 2-D sheet of FK4v simulated cardiac cells. We did this by constructing an OR gate, in a manner similar to early work by Motoike and Yoshikawa [27] using the FitzHugh-Nagumo model [46]. This was then concatenated with a NOT gate. However, the previous work had a neurological basis and we have made choices more in line with cardiac tissue behaviour.

The principal layout for our OR is a simple **Y** shape shown in Figure 2. The two top points of the Y being the inputs and the base being the output. This is implemented as normally active endocardial cells, whilst the rest of the sheet is made up of one of two abnormal cell types. As per our previous work [45], these are either diffusive or zero flux cells.

Diffusive cells are completely non-excitable whilst still allowing membrane voltage to diffuse through them, these are modelled by setting $I_{ion} = 0$ in Equation 1. Zero Flux Cells (ZFCs) allow no membrane voltage to enter them and present themselves to neighbouring cells like zero-flux boundary conditions. These can be seen as the extreme form of reduce excitability and reduced connectivity respectively, so any conclusions derived from these cell types could be extrapolated to more realistic cells with a length scaling.

In standard logic circuits, the presence of a voltage indicates 1/true, whilst the absence indicates 0/false. We choose a better fit to a cardiac model context in that we used a train of APs at an interval of 0.8 s and a base time of zero for 1/true, and a *delayed* train also at an 0.8 s interval but with a base of 0.4 s. Hence, 0 and 1 side by side would be seen to alternate.

Voltage plots against time for locations near to each input and the output, for the ZFC abnormal cells and each input regime (11, 01, 10 and 00) can be seen in Figure 4. The output for the diffusive abnormal cells being essentially the same.

In the 11 and 00 cases the synchronous signals merely pass through the system unchanged, whilst in the 01 and 10 case the 1 travels back up the other input and blocks it and also continues on to produce the output. Clearly, if we take a base time of ~ 0.2 s at the output we obtain the necessary output for an OR gate.

Using this half interval time delay to differential between 0 and 1 allows us to create a NOT gate by simply sending our conductive track on a detour that takes the AP the additional half interval to navigate.

One problem of this run back to cancel out the 0 if this gate were to be used in a wider logic circuit, is that it would continue into previous gates and interfere with their function. This can be rectified easily in the diffusive cell case by implementing a diode after each logic gate. By diode we mean some form of structure which only allows the passage of an AP in one direction. Such a structure with diode function has already been constructed in other RD media [27, 47, 48], and was easily transferred to the FK4v cardiac model with diffusive cells. See Figure 5 for a systematic diagram, and Figure 6 for snapshots of the system in action.

It is not clear as to how to carry out the same function with ZFCs, but it must be kept in mind that they are the limiting case and we would expect *real* reduced conductivity cells to have a reduced diffusion coefficient not zero. As such the equivalent diode structure can also be constructed with a reduced diffusion coefficient abnormal cells.

VII. SPEED COMPARISON OF CPU & GPGPU

GPGPU techniques are often suggested to yield great computational efficiency gains, to get a simple measure of this in our case we implemented our previous cardiac simulation code on the XBox as both a standard CPU and as a GPGPU programme.

In the CPU case we essentially cut and paste our old code into a stand alone C^{++} class which was then implement in a base code used at Rare to develop prototypes and technical demonstrations. This loaded the simulation input data, built the system, ticked the simulation and interfaced with it to produce both the standard and graphical output. Importantly, we just changed enough to get the original code to work within the framework, and no optimization was carried out to take advantage of running on the XBox.

In as similar manner as possible our GPGPU code was also set up within the same framework, with the primary difference being the simulation and the production of the graphical output being carried out exclusively on the GPU.

To compare the two programmes we carried out the obvious comparison and ran each code with a series of progressively larger squares of simulated tissue, and calculated the number of iterations per second in each case. A plot of this can be seen in Figure 3. This shows that even with a small 10 X 10 square, the GPGPU simulation is more than twice as fast as the CPU. The CPU simulation rapidly drops to 14 iterations per second, whilst at 200 X 200 the GPGPU is still at 354 iteration per second.

As previously stated the CPU code was not optimized for running on the XBox, but the original code was always designed more for ease of expansion and use for multiple types of model and system, than for speed. So optimizing the code could yield a marked speed increase. However, the GPU code was also not optimized, being essentially the first pass at getting the simulation in a GPGPU form. So optimization could make both codes far faster, but it is highly unlikely that the CPU code could gain the needed orders of magnitude to beat the GPGPU code as it currently stands, let alone after it was also optimized.

The speed gains here for our GPGPU version allowed us a very much more "handson" approach to producing structures to be simulated. The program was run with given input, the simulation could be observed running if not in precisely *real-time* then in a user comfortable amount of time, the input was updated in light of the simulation and the process repeated. This allowed for very rapid development of the required structures for the simulations here presented, certainly at a speed impossible with the previous CPU based code framework even when running in a parallel form.

Of course these gains in speed were heavily predicated on us using the special features of the XBox 360 available to professional games developers. The memory export function for example is disabled for security reasons in the publicly available XNA XBox development environment. However, it is to be expected that near equivalent gains could have been made using a traditional texture based GPGPU approach and further that as interest in GPGPU increases with the sophistication of GPUs that such improved memory access will become standard for GPUs and their APIs.

One final benefit of GPGPU techniques is in the area of visualization. If one wishes to do 3-D visualization for a general simulation on modern graphics hardware one must convert your simulation data in to a GPU useable form *i.e.* vertices and textures. However, if you are using GPGPU then this has already been done, and essentially "free" visualization can be achieved by simply tacking onto the end of the simulation shaders the required calculations for visualization. Although not shown here, we produced a number of such visualization shaders. For example, one which used the v, w and d parameters of the FK4v model and mapped them to the red, green and blue colours of the displayed tissue, allowing easy observation of the evolution of the simulation parameters in "real-time". We also produced a pixel shader which highlighted phase singularities, so called filaments. The structure and dynamics of which being an important area of study in cardiac simulations.

VIII. PREDICTION OF RE-ENTRY

Interestingly, much of the previous work on producing computational structures in RD systems concentrated on neurological models and drew conclusions on such systems informed by computational concepts, as shown by many of the references in this work. These in general suggested links between computational functionality between neurons and electronics. We shall now draw higher level conclusions for cardiac systems based on the theoretical basis of computation itself rather than specific computational functions / structures.

Something of a Holy Grail in the field of cardiac electrophysiology would be a technique which can observe an arbitrary piece of heart tissue, pin point the location of where tissue behaviour is altered by disease or injury, and then states whether an electrical excitation would be both re-entrant and self-sustaining. However, in so far as this would require analysis of APs interacting with diseased cell structures within a cardiac model, this amounts to asking if the program of a Turing-machine does not halt. Reversing the Halting problem in this manner does nothing to render it any less impossible so such a technique is a logical impossibility.

Although, one can play a *get out of gaol free* card in that long term re-entry sufficient to cause illness or death may not be the same as infinitely self-sustained. As such behaviour merely needs to continue for a given finite period to produce ill effects/death. This does not fall foul of the halting problem as the question "does this program keep running for a given finite time" is quite possible. At the most basic level one just runs the program for the given time and observes if it terminates. Although, the impossibility tends to leak into the possible side, as the infinite form is merely a limiting case of the finite. Therefore, any solution past this trivial one is likely to be extremely difficult.

This Turing completeness even suggests the possibility that cardiac cells could be used as the basis of a bio-mechanical computing device, using either techniques close to that of a TRDM or more prosaically using advanced cell patterning techniques, for example directed cell growth using micro-contact printing of adhesive proteins [49], and essentially grow the required logic circuit from heart tissue.

IX. CONCLUSION

In this work we have constructed the universal logic gate NOR using simulated cardiac cells of two base defective types, clearly showing that diseased cells in simulated cardiac tissue interacting with APs can be considered a Turing complete system. From this we highlight that a methodology for deducing whether a given arrangement of abnormal tissue produces self-sustaining re-entry is therefore a logical impossibility as it equates to a solution of the Halting problem.

We have further shown that GPGPU is a highly effective way of carrying out high end parallel computing on "domestic" hardware for cardiac simulations. Although major reworking of any previous code framework is required, this cost can easily be out-wayed by the benefits in gained computational power and speed, as well as the relative ease of visualization of the system.

Acknowledgments

We would like to thank our current colleagues Tom Grove and Sebastian Sylvan of Rare Ltd. (www.rareware.com) for assistance with the GPGPU aspects of this work. We would also like to thank our previous colleagues Richard Clayton and Susheel Varma of the University of Sheffield for assistance with the cardiac aspects.

[1] D. P. Zipes and H. J. Wellers. Sudden Cardiac Death. *Circulation*, 98:2334–51, 1998.

[2] M. J. Davies. Pathological view of sudden cardiac death. *Br. Heart J.*, 45:88–96, 1981.

[3] S. Goldstein, J. R. Londis, R. Leighton, G. Ritter, C. M. Vasu, and A. Lantis. Characteristics of the resuscitated out-of-hospital cardiac arrest victim with coronary heart disease. *Circulation*, 64:977–984, 1981.

[4] A. Turing. The Chemical Basis of Morphogenesis. *Philos. Trans. of the Royal Soc. of London Series B*, 237:631, 1952.

[5] http://www.gpgpu.org.

[6] R. H. Clayton. Computational models of normal and abnormal action potential propagation in cardiac tissue: Linking experimental and clinical cardiology. *Physiological Measurement*, 22:R15–R34, 2001.

[7] A. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proc. of the London Math. Soc.*, 42:230–265, 1936.

[8] L. Kuhnert. Photochemische Manipulation von chemischen Wellen. *Naturwissenschaften*, 76:96–7, 1986.

[9] L. Kuhnert, K. L. Agladze, and V. I. Krinsky. Image processing using light-sensitive chemical waves. *Nature*, 337:244–7, 1989.

[10] A. Adamatzky. Collision-based computing in Belousov-Zhabotinsky medium. *Chaos, Solitons and Fractals*, 21:1259–64, 2004.

[11] A. Adamatzky. *Computing in Nonlinear Media and Automata Collectives.* IoP, London, 2001.

[12] A. Adamatzky. Computing with waves in chemical media: massively parallel reaction-diffusion processors. *IEICE Trans*, 11:1748–56, 2004.

[13] A. Adamatzky. Programming Reaction-Diffusion Processors. *Lec. Notes in Comp. Sci.*, 356:33–46, 2005.

[14] N. G. Rambidi. *Chemical-based computing and problems of high computational complexity: The reaction-diffusion paradigm. In: Molecular Computing.* The MIT Press, USA, 2003.

[15] A. Adamatzky and B. P. J. De Lacy Costello. Collision-free path planning in the Belousov-Zhabotinsky medium assisted by a cellular automaton. *Naturwissenschaften*, 89:474–8, 2002.

[16] K. Agladze, N. Magome, R. Aliev, T. Yamaguchi, and K. Yoshikawa. Finding the optimal path with the aid of chemical wave. *Physica D*, 106:247–54, 1997.

[17] N. G. Rambidi and D. Rakovenchuck. Finding path in a labyrinth based on reaction-diffusion media. *Adv. Mater. Opt. Electron.*, 7:67–72, 1999.

[18] O. Steinbock, Á. Tóth, and K. Showalter. Navigating complex labyrinths: optimal paths from chemical waves. *Science*, 267:868–71, 1995.

[19] A. Adamatzky, P. Arena, A. Basile, R. Carmona-Galan, B. De Lacy Costello, L Fortuna, and et al. Reaction-diffusion navigation robot control: from chemical to VLSI analogic processors. *IEEE T Circuits Syst-I*, 51:926–38, 2004.

[20] A. Adamatzky, B. De Lacy Costello, C. Mehuish, and N. Ratcliffe. Experimental implementation of mobile robot taxis with on board Belousov-Zhabotinsky chemical medium. *Mat Sci Eng C*, 24:541–8, 2004.

[21] H. Yamada T. Nakagaki and Á. Tóth. Intelligence: Maze solving by an amoeboid organism. *Nature*, 407:470, 2000.

[22] T. Nakagaki. Smart behaviour of true slime mould in a Labyrinth. *Res. Microbol.*, 152:767–70, 2001.

[23] Á. Tóth and K. Showalter. Logic gates in excitable media. *J. Chem. Phys.*, 103:2058–66, 1995.

[24] T. Ichino, Y. Igarashi, I. N. Motoike, and K. Yoshikawa. Different operations on a single circuit: Field computation on an excitable chemical system. *J. Chem. Phys.*, 118:8185–90,

2003.

[25] I. N. Motoike and K. Yoshikawa. Information operations with multiple pulses on an excitable field. *Chaos, Solitons and Fractals*, 17:455–61, 2003.

[26] J. Sielewiesiuk and J. Górecki. On the response of simple reactors to regular trains of impulses. *Phys. Chem. Chem. Phys*, 4:1326–33, 2002.

[27] I. Motoike and K. Yoshikawa. Information operations with an excitable field. *Phys. Rev. E*, 59:5354–60, 1999.

[28] J. Gorecki, K. Yoshikawa, and Y. Igarashi. On Chemical Reactors That Count. *J. Phys. Chem. A*, 107:1664–9, 2003.

[29] I. N. Motoike, K. Yoshikawa, Y. Iguchi, and S. Nakata. Real-time memory on an excitable field. *Phys. Rev. E*, 63:036220, 2001.

[30] C. Simone and S. Bandini. The reaction-diffusion metaphor for modelling cooperative work. *Prestige J. of Management and Research*, 2:1–21, 1998.

[31] S. Bandini and C. Simone. Integrating Form of Interaction in a Distributed Model. *Fundamenta Informaticae*, 61:1–17, 2004.

[32] S. Bandini, G. Mauri, G. Pavesi, and C. Simone. Computing with a Distributed Reaction-Diffusion Model. *Lec. Notes in Comp. Sci.*, 3354:93–103, 2005.

[33] *Crackdown.* Realtime Worlds, Microsoft Game Studios, 2007.

[34] *Okami.* Capcom, 2006.

[35] *Prince of Persia.* Ubisoft, To be published.

[36] A. Lake. Cartoon Rendering Using Texture Mapping and Programmable Vertex Shaders. *Game Programming Gems*, 2:444–51, 2001.

[37] B. Freudenberg, M. Masuch, and T. Strothotte. Real-Time Halftoning: Fast and Simple Stylized Shading. *Game Programming Gems*, 4:443–9, 2004.

[38] M. Harris. Mapping Computational Concepts to GPUs. *GPU Gems*, 2:Chapter 31, 2005.

[39] J. D. Owens et al. A Survey of General-Purpose Computation on Graphics Hardware. *Europhysics 2005*, State of the Art Reports:21–51, 2005.

[40] W. Liu, B. Schmidt, G. Voss, and W. Muller-Wittig. Accelerating molecular dynamics simulations using Graphics Processing Units with CUDA. *Computer Physics Communications*, 179:634–641, 2008.

[41] P. Richmond and S. Coakley. A High Performance Agent Based Modelling Framework on

Graphics Card Hardware with CUDA. *Proc. of 8th Int. Conf. on Autonomous Agents and Multi-agent systems*, to be published.

[42] M. Nagle. Games Consoles Reveal their Hidden Power. *New Scientist*, 2643:26–7, 2008.

[43] D. Robson. From CPU to GPU. *HPC for Science*, 1:8–10, 2008.

[44] F. Fenton and A. Karma. Vortex dynamics in three-dimensional continuous myocardium with fibre rotation: Filament instability and fibrillation. *Chaos*, 8:20–47, 1998.

[45] S. Scarle and R. H. Clayton. Initiation of re-entry in an excitable medium: A Structural investigation of cardiac tissue using a genetic algorithm. *Chaos*, 16:03315, 2006.

[46] R. FitzHugh. *Biophysics. J.*, 1:1961, 2008.

[47] T. Yamaguchi K. Agladze, R. Aliev and K. Yoshikawa. Chemical Diode. *J. Phys. Chem.*, 100:13895–7, 1996.

[48] T. Yamaguchi *et al.* Unidirectionality of Chemical Diodes. *ACH-Models in Chem.*, 135:401, 1998.

[49] J. Tan et al. Simple approach to micro-pattern cells on common culture substrates by tuning wettability. *Tissue Eng.*, 10:865–72, 2004.

Gate	input		Output
	1	2	
AND	1	1	1
	1	0	0
	0	1	0
	0	0	0
OR	1	1	1
	1	0	1
	0	1	1
	0	0	0
NOT	1	-	0
	0	-	1
XOR	1	1	0
	1	0	1
	0	1	1
	0	0	0

TABLE I: The standard logic gates.

J	description	ion
fi	fast inward	Na^+
so	slow outward	K^+
si	slow inward	Ca^+

TABLE II: Description of ionic currents used in the Fenton Karma 4 variable model.

green value	description
0	set V_m to blue
1	add blue to V_m
2	add blue percent to V_m
3	reset all cells values to starting values

TABLE III: Types of stimulation that can be applied by the Zapp shader.

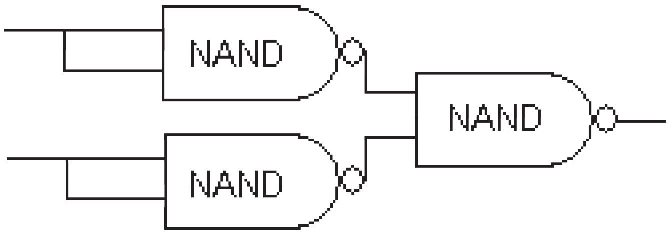

FIG. 1: Example of the universality of the NAND gate, the function of an OR gate produced from a set of NAND gates.

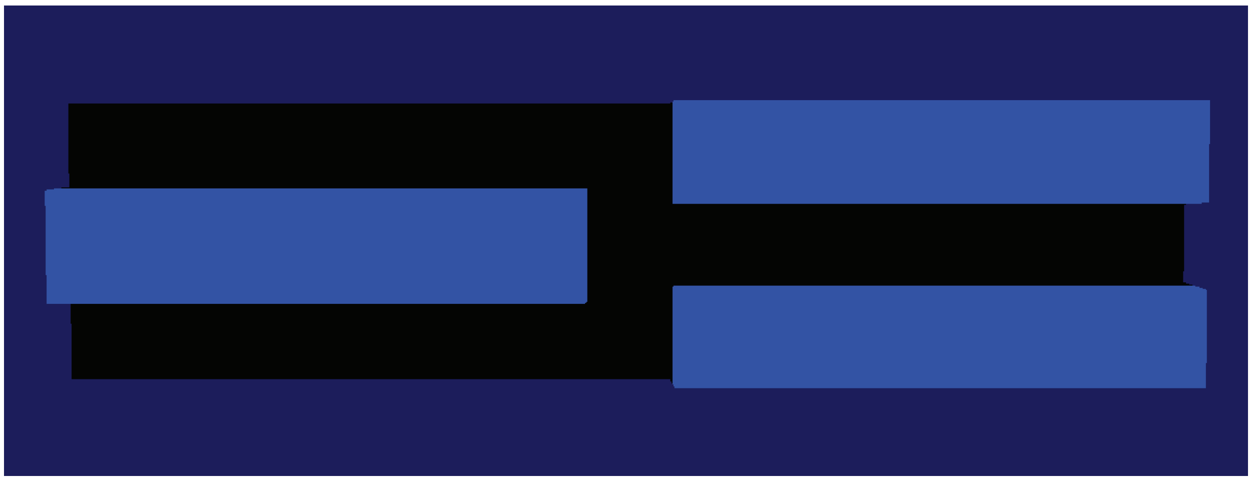

FIG. 2: Screen capture from the XBox 360 showing the basic structure of our OR gate. Blue being the abnormal cells and black being normal endocardial cells.

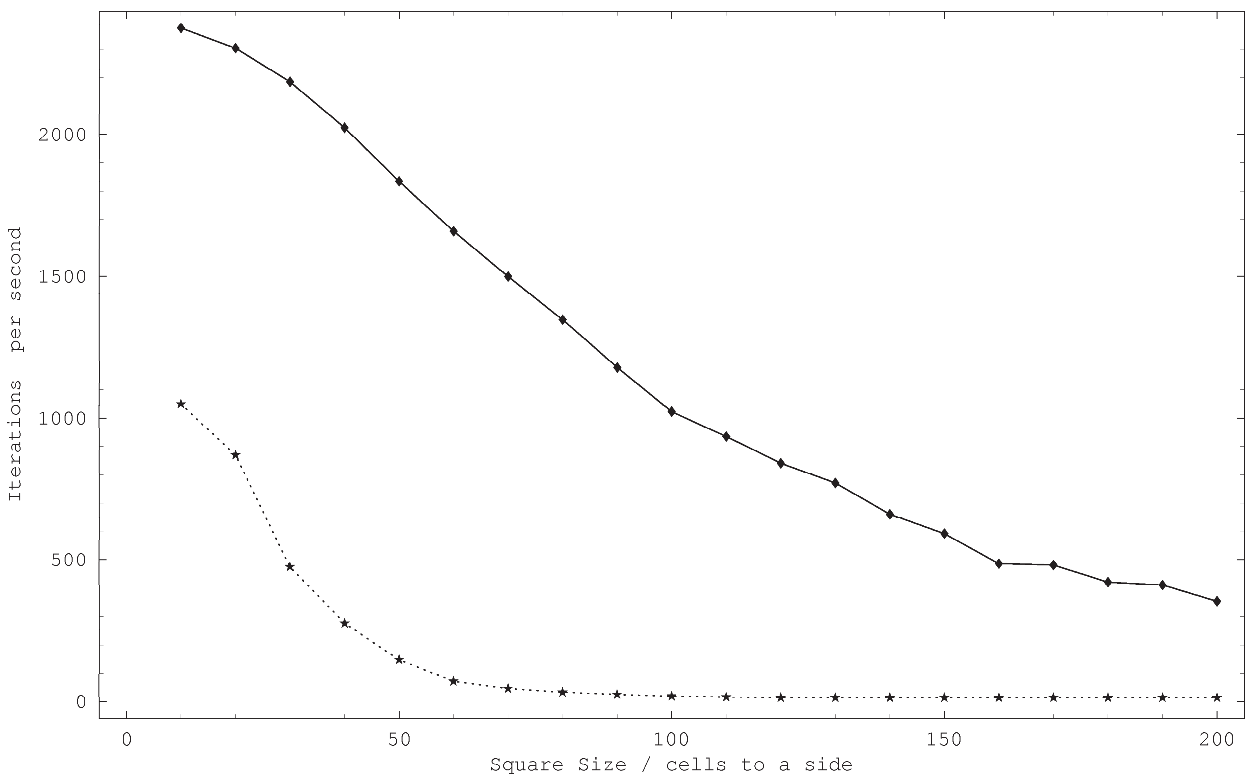

FIG. 3: Plot comparing speed of simulation in frames/iterations per second for varying sizes of simulated tissue square for the CPU and GPGPU implementations. Broken line being CPU and solid line being GPU.

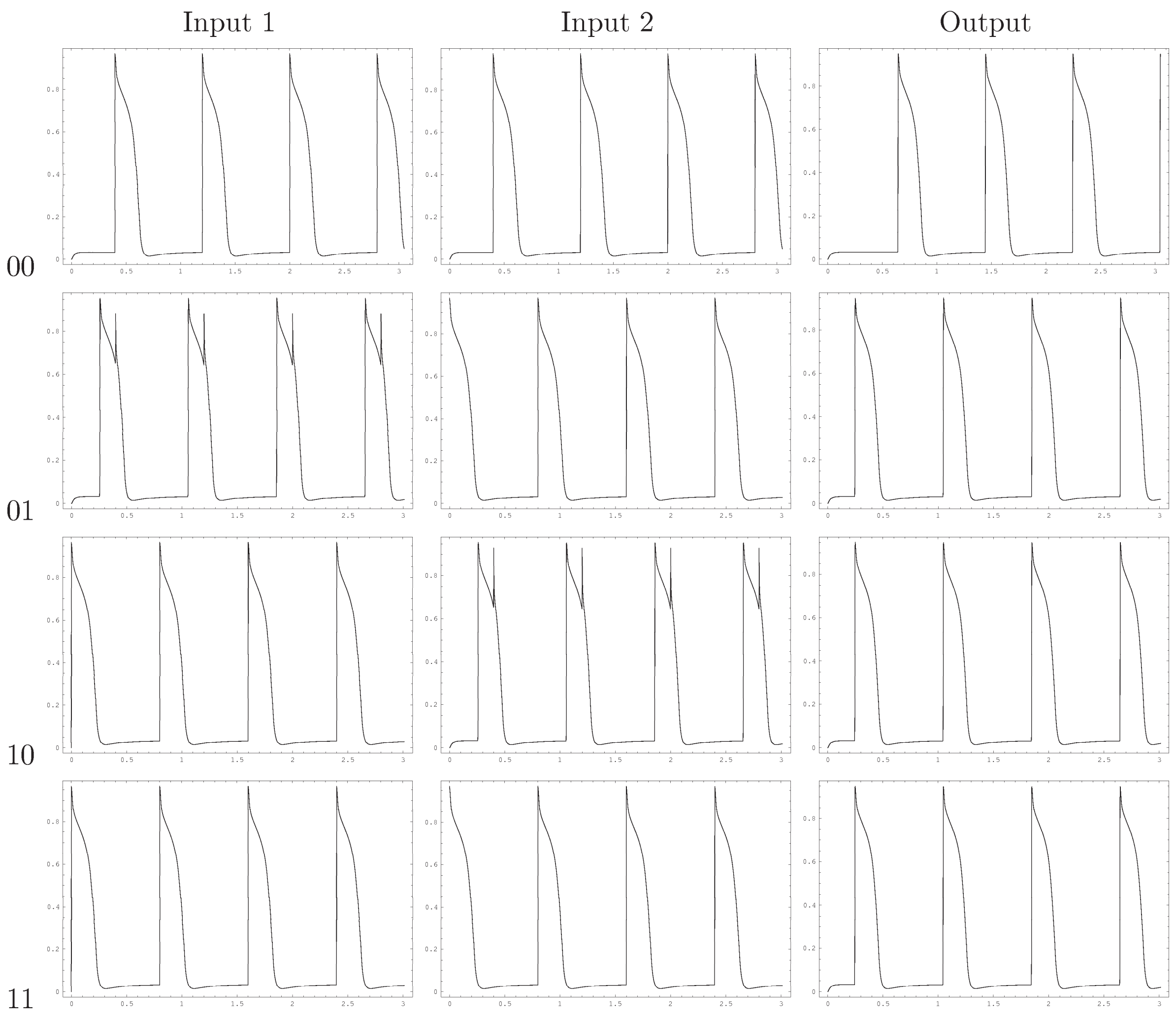

FIG. 4: Plots of voltage against time for each input regime and the ZFC abnormal cell type. Abscissa being membrane voltage in arbitrary units and the ordinate being the simulated time in seconds.

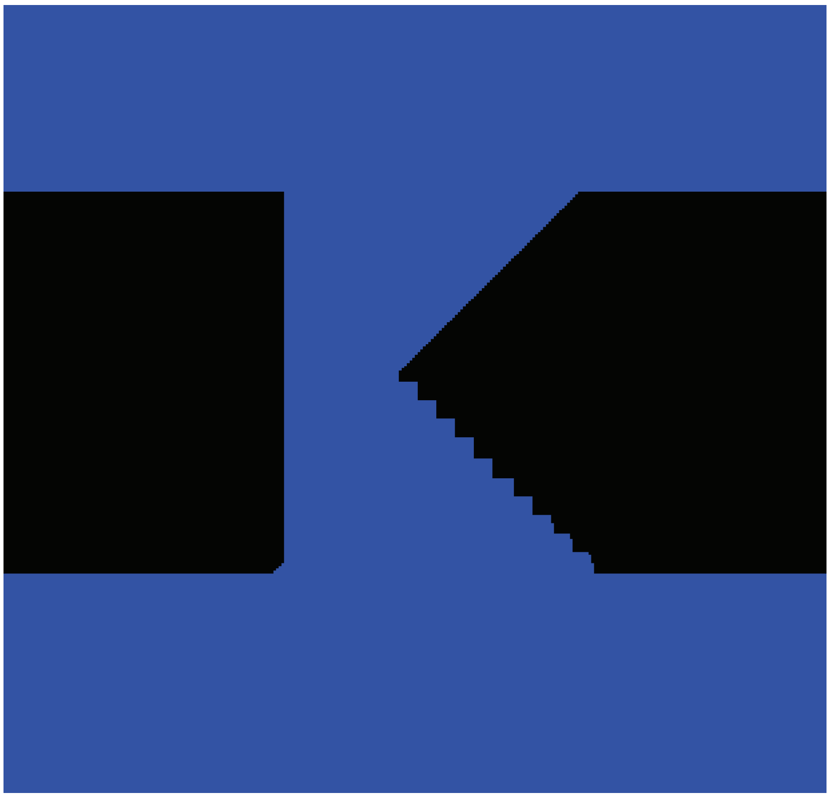

FIG. 5: A detail of a screen shot from the XBox 360 showing diode structure. Blue being the diffusive abnormal cells and black being normal endocardial cells.

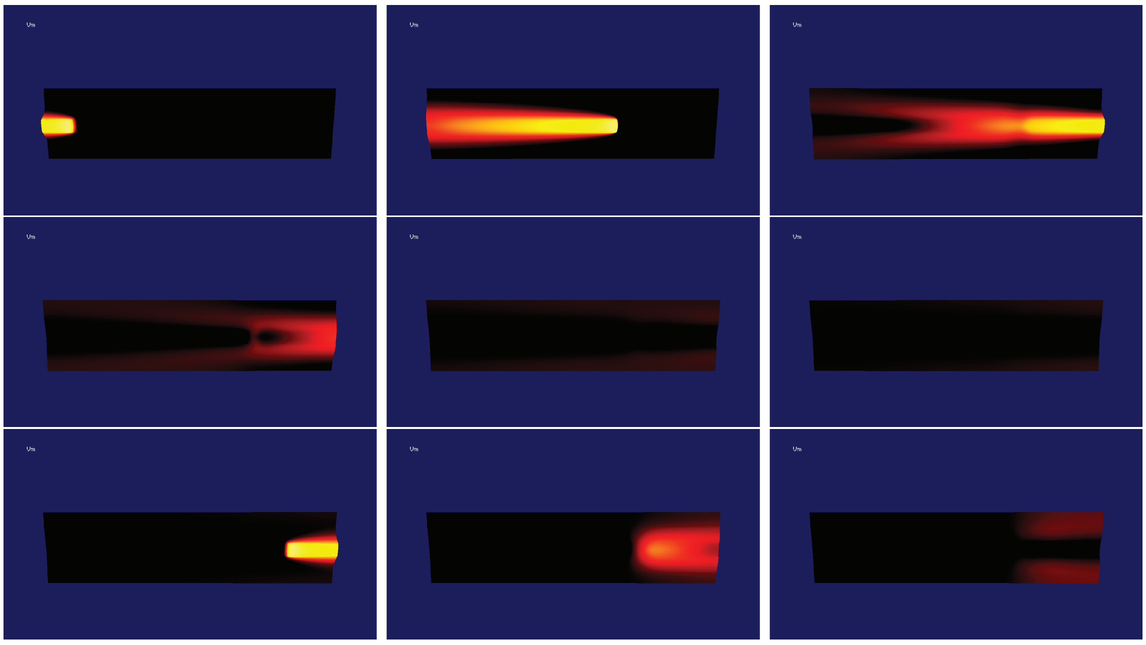

FIG. 6: Snapshots of the diode in action, showing and outward AP passing through and the block of a returning AP.