COMPUTER SCIENCE

Real-time interactive simulations of large-scale systems on personal computers and cell phones: Toward patient-specific heart modeling and other applications

Abouzar Kaboudian[1], Elizabeth M. Cherry[2], Flavio H. Fenton[1]*

Cardiac dynamics modeling has been useful for studying and treating arrhythmias. However, it is a multiscale problem requiring the solution of billions of differential equations describing the complex electrophysiology of interconnected cells. Therefore, large-scale cardiac modeling has been limited to groups with access to supercomputers and clusters. Many areas of computational science face similar problems where computational costs are too high for personal computers so that supercomputers or clusters currently are necessary. Here, we introduce a new approach that makes high-performance simulation of cardiac dynamics and other large-scale systems like fluid flow and crystal growth accessible to virtually anyone with a modest computer. For cardiac dynamics, this approach will allow not only scientists and students but also physicians to use physiologically accurate modeling and simulation tools that are interactive in real time, thereby making diagnostics, research, and education available to a broader audience and pushing the boundaries of cardiac science.

INTRODUCTION

Heart disease remains the leading cause of death worldwide (*1*), with fatal cardiac disease often associated with spatiotemporal disorganization of the normal electrical signal that drives the ventricles' contraction (*2*–*4*). This disruption can require immediate intervention, as with ventricular fibrillation (VF), or, as with atrial fibrillation (AF), may last for years with impaired quality of life and increased risk for other cardiac diseases like stroke. VF generally can be treated by expensive implantation of a cardioverter defibrillator in at-risk patients. AF has no widely effective long-lasting treatment option; for example, catheter ablation to interrupt repetitive abnormal electrical activity is not effective for all patients and often requires follow-up treatments (*5*). It may be possible to improve outcomes in both cases by designing patient-specific prevention, control, or therapy. However, the advancement and widespread adoption of these approaches requires new computational tools that are fast, accessible, and easy to use.

Personalized treatment tools are likely to use individualized cardiac anatomies populated with mathematical representations of cardiac cells. Numerous mathematical cardiomyocyte models based on ion channel currents have been developed (*6*); they have helped in understanding arrhythmia mechanisms (*7*), designing methods for control and defibrillation (*8*), and studying proarrhythmic and antiarrhythmic drug effects (*9*). Thus, numerical simulations of cardiac dynamics are becoming increasingly important in addressing patient-specific interventions (*10*) and evaluating drug effects (*11*). The U.S. Food and Drug Administration (FDA) recently sponsored a new Cardiac Safety Research Consortium initiative [Comprehensive In Vitro Proarrhythmia Assay (CiPA)] (*11*, *12*) that specifies the use of mathematical cardiac models to aid proarrhythmic drug risk assessment. A complicating factor is that mathematical models have become extremely complex, with some needing 50 to 100 complex differential equations to account for all the processes of a cell (*6*), leading to two main problems. First, these models require substantial expertise to run even without considering behavior that arises through intercellular coupling, in 2D (two-dimensional) and 3D; only a handful of groups in the world have the necessary coding expertise and access to supercomputers to run complex cell models in 2D and 3D. Second, understanding the roles of the many variables and parameters used in these models, which is necessary to develop and validate personalized models, requires extensive and time-consuming parameter sensitivity studies and uncertainty quantification analysis (*13*).

Although our main interest is in cardiac modeling, the high computational cost of modern numerical simulations is not limited to cardiac simulations. Various different fields such as fluid mechanics, elastic solid mechanics, fluid-solid interaction problems, geophysical modeling, and even astrophysical simulations impose huge computational demands that are currently addressed through utilization of supercomputers.

To tackle these problems and make progress in producing tools useful for computer-aided therapy planning, large-scale parallelization is necessary. The current hardware solution to significantly increase computational bandwidth is to use graphics processing units (GPUs). A typical central processing unit (CPU) can solve about 10^8 ordinary differential equations (ODEs) per second, whereas a modern consumer-level GPU through parallelization can solve 3×10^{10} to 4×10^{10} ODEs per second. However, the development and maintenance of codes that efficiently use GPU resources are currently challenging: Specialized knowledge of GPU architecture is required for maximum benefit, and coding specifications may change depending on the operating system or GPU device used.

To overcome many of these challenges, we have developed a fast simple library using Web Graphics Library (WebGL 2.0), which is a combination of a JavaScript Application Programming Interface and GLSL (OpenGL Shading Language with a syntax similar to C-C++). WebGL codes execute in parallel on the GPU and run interactively through HTML-5 canvas element on any modern web browser. WebGL 2.0 and JavaScript are, by design, independent of the device and operating system and do not require any plugins or toolkits. Therefore, recompilation of software under WebGL and JavaScript is not needed, even when switching operating systems or hardware. Instead, programs are broadly

[1]School of Physics, Georgia Institute of Technology, Atlanta, GA, USA. [2]School of Mathematical Sciences, Rochester Institute of Technology, Rochester, NY, USA.
*Corresponding author. Email: flavio.fenton@physics.gatech.edu

accessible and easy to maintain: They can be downloaded from a website and run locally by simply clicking on their web link. Furthermore, visualization and interactivity are directly included at run time.

Here, we propose a significant step toward achieving the ability of personalized computing for the treatment of cardiac disease by (i) harnessing the power of GPUs for high-performance scientific computing via WebGL and (ii) developing a fast specialized library (`Abubu.js`) for efficient simulations of complex partial differential equations that model complex cardiac cell models in tissue, including physiologically accurate simulations on ventricular and atrial structures. (iii) We validate these tools with near–real-time simulations of models that quantitatively reproduce experimental data. (iv) We further show the versatility of the library and WebGL codes by applying it to other computationally expensive problems that are not related to cardiac dynamics such as crystal growth and fluid flow.

MODELS

Modeling cardiac electrophysiology from single cells to tissue

The electrical dynamics of cardiac cells is typically modeled by using ODEs to describe the various ionic currents (*6*) that produce the cell membrane's change in voltage, called an action potential (AP), that triggers the release of intracellular calcium, leading to the cell's contraction. In tissue, the voltage is modeled by a nonlinear reaction-diffusion equation (*14*) known as the cable equation

$$\frac{\partial V}{\partial t} = \nabla.(\mathbf{D}\nabla V) - \Sigma J_{\text{ion}}$$

where V is the transmembrane potential, $\mathbf{D}$ is the diffusion tensor that contains the tissue's structure and rotational anisotropy (*2*, *14*), and ΣJ_{ion} indicates the sum over all ionic currents for the cell (*6*). This equation assumes that the extracellular tissue is grounded, an approximation that holds for most studies of cardiac dynamics except for those that require extracellular effects such as defibrillation studies; in these cases, a bi- or tridomain model is required (*8*, *15*).

Most models use a Hodgkin-Huxley approach to model ionic currents, where the current density follows Ohm's law but with the conductance a highly nonlinear function. The current through each ion channel is determined, in part, by one or more gating variables of the form

$$\frac{dy}{dt} = \frac{y_\infty(V) - y}{\tau_y(V)}$$

where y_∞ is the voltage-dependent steady-state value of the gate y and τ_y is the voltage-dependent (activation or inactivation) time constant of the gate. Some ion channels open and close in response to other factors as well, such as intracellular Ca^{2+} or extracellular K^+ concentration. Some models use a Markov chain approach to model some of the ion currents by using discrete states representing various configurations of the channel along with allowable transitions. Models also include pumps and exchangers to model ion transport across the membrane by active processes rather than simple diffusion and complex intracellular calcium handling that accounts for calcium released from the sarcoplasmic reticulum (SR), ion diffusion within the SR and cytoplasm, and reuptake of calcium back into the SR.

The number of variables and ODEs required for a particular model in a single cell depends on the number of detailed ionic currents, pumps, exchangers, and ion concentrations used (*6*). In this work, we use some of the most popular models, including a 4-variable minimal model (MM) (*16*) and two human ventricular cell models, the 19-variable ten Tusscher-Panfilov (TP) model (*17*) and the 41-variable O'Hara *et al.* (OVVR) model (*18*), to illustrate how it is possible to simulate and interact in real time with complex models in 2D and 3D. Solving these models in real time in 2D tissues can require as many as 10 to 100 billion ODEs for 1 s of simulation; using a 3D heart structure can require solving 200 times more equations per second, which is several orders of magnitude greater than the speeds possible for current CPU-based computers.

Modeling other fields: Fluid flow and crystal growth

The exorbitant computational costs for modeling physical systems are not specific to the field of cardiac dynamics. For example, fluid flow around or between obstacles is a common phenomenon in applications that range from offshore oil and gas risers (*19*), wind turbines (*20*), airplanes, civil structures, and cooling towers in thermal power plants (*21*) to small-scale problems such as blood flow in vessels, flow in porous media, and many more. In external flows, vortex shedding subjects structures to cyclic loading that, in turn, can lead to fatigue problems in the structures. Fatigue reduces the life of structures significantly, leads to structural failure, and can have significant financial burden and fatal consequences with huge environmental impacts in some applications such as offshore oil and gas. Hence, simulations at the design stage can help facilitate suppression or minimization of such cyclic loading. However, these simulations usually require massive computations due to small length scales either in fluid flow or in the structures that demand a high spatial resolution as well as stiff differential equations that require small temporal resolutions and thus the use of supercomputers.

Hydrate and crystal formation and dissolution is another field that has broad applications in geophysical studies and metallurgy with uneven solidification of solids, among others. The phase-transition phenomena that happen in the presence of fluid flow have extra layers of complexity that also often require the use of supercomputers.

These problems can be solved, for example, by using a lattice Boltzmann method (LBM), which can be easily parallelized (*22*). While the LBM formulations can benefit significantly from parallelization, they still require a parallel platform. Traditionally, supercomputers have been the platform of choice for the LBM methods. In this work, we have also used our developed library `Abubu.js` to implement the LBM formulations for the fluid flow problem and the crystal growth problem in WebGL.

METHODS

Numerical methods

While there have been efforts for creating interactive simulations of cardiac and excitable models, they have been mostly done for relatively simpler models (*23*, *24*), so traditionally, complex multidimensional simulations of cardiac dynamics as well as other computationally costly models have been carried out using large supercomputers, but these resources are expensive to acquire and maintain and are difficult for nonspecialists to use. GPUs, a recent alternative to CPU computing, solve some of these problems by providing a low-cost alternative. GPUs provide thousands of computational cores that can carry out mathematical operations in parallel. In this way, they provide high-performance computing at the personal device level.

However, programming GPUs for optimal performance presents new challenges by requiring specialized knowledge and techniques that

vary with different operating systems and GPU hardware, making development and maintenance of codes difficult. Several languages exist to develop programs for GPUs (*25*) and several implementations, particularly CUDA, have allowed accelerations of simulations in tissue (*26*) and for several complex models (*27*); however, the codes need to be compiled and optimized for particular architectures (they are executable only on NVIDIA graphic cards). Here, we provide an alternative through `Abubu.js` to simplify developing computational codes that are cross-platform, do not require explicit compilation by developers or users, and can be easily accessed and executed simply by visiting a webpage. We further show examples that enable simulations to run several orders of magnitude faster on personal computer (PC) GPUs.

Developing WebGL computational codes using `Abubu.js`

WebGL 2.0 is a relatively low-level application program interface (API) developed to display 2D and 3D graphics in a modern web browser. Hence, using it to carry out numerical simulations can be quite daunting for programmers who might not be well versed in graphics card programming. In this work, we have developed a library, `Abubu.js`, that removes most of the complexities involved in dealing with the graphical aspect of the programming and instead allows users to focus on developing numerical programs that can easily run in a modern web browser and harness the immense power of the GPU. Furthermore, by default, simulation results can be directly plotted on the screen, thereby directly integrating visualization and interaction with the computation.

In this section, we briefly review the programming process for implementing a model in WebGL using the `Abubu.js` library for an example cardiac model. For this example, we have developed an MM (consisting of three variables) to describe porcine cardiac electrophysiology (see the "Experimental methods" section). Therefore, the description below serves two purposes: to present the equations of a new model for porcine ventricular cells and to show how to implement it in WebGL for simulations in 2D and 3D using our library. The general idea behind `Abubu.js` is the use of rectangular images, otherwise known as textures, as the primary data structure. Each image naturally contains a grid of pixels, and each pixel contains four color channel values, namely, red, green, blue, and alpha. In our paradigm, by assigning a physical variable to each color channel, we can treat each pixel of an image as a numerical grid point. While this is not the first time to use images as data structures (*28*), our library facilitates the use of these data structures as input and output so that programmers who are not experts in the graphical pipeline design can easily start implementing the numerical models with minimal effort. Furthermore, our library allows easy output to multiple textures to facilitate programming models with tens of variables per point in space.

To further clarify this step, consider the following three-variable MM of porcine ventricles, which follows a formulation similar to the Hodgkin-Huxley model of a neural membrane potential (*29*)

$$\frac{\partial u}{\partial t} = \nabla.(\mathbf{D}\nabla u) - (I_{\mathrm{fi}} + I_{\mathrm{si}} + I_{\mathrm{so}})/C_{\mathrm{m}}$$

where u is the normalized transmembrane potential; $\mathbf{D}$ is the diffusion tensor describing tissue structure; I_{fi}, I_{si}, and I_{so} are the fast inward, slow inward, and slow outward ionic currents that roughly equate to a total sodium current, a total calcium current, and a total potassium current; and C_{m} is the membrane capacitance. The currents are given by

$$I_{\mathrm{fi}} = -\frac{vp(u-0.1)(0.97-u)}{0.175}$$

$$I_{\mathrm{si}} = -w\frac{\{1.0-\tanh[10.0(u-0.9)]\}\{1.0+\tanh[7.0(u-0.35)]\}}{62.0\{1.0+\exp[4.5(u-0.9)]\}}$$

$$I_{\mathrm{so}} = \frac{u(1.0-p)(1.0-v)}{4.5} + \frac{p}{5.0+15.0\{1.0-\tanh[50.0(u-0.85)]\}}$$

where v and w are sodium and calcium gating variables that are governed by

$$\frac{dv}{dt} = \frac{(1.0-p)(1.0-v)}{40.0q+(1.0-q)2000.0} - \frac{pv}{10.0}$$

$$\frac{dw}{dt} = \frac{(1.0-p)(1.0-w^4)}{305.0} - \frac{pw}{320.0}$$

p and q are thresholding variables used to define the step functions in the model and are calculated by

$$p = H(u-0.25)$$
$$q = H(0.0025-u)$$

Here, H is the Heaviside function defined to be 1 if its argument is nonnegative and 0 otherwise. The procedure for parametrization and validation of this model follows in the next sections.

2D Implementations using `Abubu.js`

Assuming a 512 × 512 2D numerical grid, we can use the following utility function to define two textures/images for time-stepping the solution.

```
var fuvw = new Abubu.Float32Texture(512,512) ;
var suvw = new Abubu.Float32Texture(512,512) ;
```

Because the codes that use these textures as input and output are massively parallel, to avoid certain shared memory parallelization problems such as competition for data, WebGL does not allow any texture to be used as both the input and the output of a WebGL program at the same time. Hence, in each particular time step, when `fuvw` is the input texture, `suvw` is the output texture, and neither is both the input and the output at the same time. However, it is possible to switch their roles in a subsequent time step to facilitate time stepping.

At the heart of a numerical WebGL code are fragment shader codes. Fragment shaders are the part of the graphical pipeline in charge of coloring every pixel/fragment on the surface of a geometry. The programmer writes a single series of instructions for coloring all the pixels. The WebGL program launches this series of instructions in parallel with all the available resources (computational cores in the GPU) and colors batches of pixels at the same time, which results in a massively parallel code. The details of launching and decomposing the domain into batches are hidden from the programmer, which significantly simplifies the parallel programming of the numerical models. This philosophy is in line with that of the Single Program, Multiple Data (SPMD) paradigm.

The shaders are programmed in GLSL, which is a C-like language with some additional features and limitations compared to C, as the codes are to run on the GPU. A quick reference for WebGL 2.0 and the GLSL language has been released by the Khronos Group, which can be found at the khronos.org website.

We note that *u*, *v*, and *w* are the only state variables of this model. Subsequently, we can start implementing the WebGL code for this model by assigning the *u*, *v*, and *w* variables to the red, green, and blue channels of the textures `fuvw` and `suvw`. We will use the forward Euler time-stepping scheme for all the time derivatives in the model and a second-order central difference scheme for the Laplacian operator in the equation of the voltage. For simplicity, and without loss of generality, we will assume a uniform and isotropic diffusion tensor where we do not consider fiber orientation for this example.

The corresponding GLSL fragment shader code for this model is given below.

```
/*-------------------------------------------------
 * precision of the floats and integers
 *-------------------------------------------------
 */
precision highp float;
precision highp int ;
/*-------------------------------------------------
 * Interface variables
 *-------------------------------------------------
 */
in vec2 pixPos ;                  /*  position of the pixel
                                      center on the the tex-
                                      ture. The coordinates
                                      are normalized and are
                                      (0,0) for the bottom-
                                      left corner and (1,1)
                                      for the top-right
                                      corner of the texture.
                                      The coordinates are of
                                      type vec2 and are in
                                      (x,y) format.       */

uniform sampler2D  inUvw ;        /*  input texture to the
                                      program             */
uniform float      ds_x, ds_y ;   /*  domain size in the x
                                      and y directions    */
uniform float      dt ;           /*  time-step (Delta t ) */
uniform float      diffCoef,C_m ; /*  diffusion coefficient
                                      and cell capacitance */

/*-------------------------------------------------
 * output textures of the shader
 *-------------------------------------------------
 */
layout (location = 0 ) out vec4 outUvw ;

/*=================================================
 * Main body of the shader
 *=================================================
 */
void main() {
   vec2   cc = pixPos ;
   vec2   size   = vec2(textureSize(inUvw, 0 )); /* reading
                                                    size of the
                                                    texture   */
   float width  = size.x ;        /* width of the texture    */
   float height = size.y ;        /* height of the texture   */
   float cddx   = size.x/ds_x ;   /* 1/delta_x               */
   float cddy   = size.y/ds_y ;   /* 1/delta_y               */
   cddx *= cddx ;                 /* 1/delta_x^2             */
   cddy *= cddy ;                 /* 1/delta_y^2             */

/*-------------------------------------------------
 * reading from textures
 *-------------------------------------------------
 */
   vec4  C = texture( inUvw , pixPos ) ;/* read color value
                                           of pixel        */
   float u = C.r ;                /* extract u from red
                                     channel              */
   float v = C.g ;                /* extract v from
                                     green channel        */
   float w = C.b ;                /* extract w from blue
                                     channel              */

/*-------------------------------------------------
 * unit vectors
 *-------------------------------------------------
 */
   vec2 ii = vec2(1.,0.)/vec2(width,height) ; /* x-dir unit
                                                 vector   */
   vec2 jj = vec2(0.,1.)/vec2(width,height) ; /* y-dir unit
                                                 vector   */

/*-------------------------------------------------
 * Calculating Laplacian of voltage
 *-------------------------------------------------
 */
   /* du2dt is du/dt. We initialize it with the diffusion
      term                                               */
   float du2dt =( (  texture( inUvw , cc+ii ).r
                   -  2.*u
                   +  texture( inUvw , cc-ii ).r )* cddx
                +  (  texture( inUvw , cc+jj ).r
                   -  2.*u
                   +  texture( inUvw , cc-jj ).r ) * cddy )
                                                  *diffCoef ;

/*-------------------------------------------------
 * Calculating derivatives of dv/dt and dw/dt
 *-------------------------------------------------
 */
   float  p     =   0. ;
   float  q     =   1. ;

   if   ( u >= 0.25   ) p  = 1.0 ;
   if   ( u >= 0.0025 ) q  = 0.0 ;

   float  dv2dt =   (1.0-p)*(1.0-v)/
                   (40.0*q+(1.0-q)*2000.0)
                   -p*v/10.0 ;
   float  dw2dt =   (1.0-p)*(1.0-w*w*w*w)/305.0
                   -p*w/320.0 ;

/*-------------------------------------------------
 * Calculating currents
 *-------------------------------------------------
 */
   float  Ifi   =   -v*p*(u-0.1)*(0.97-u)/0.175 ;
   float  Iso   =   u*(1.0-p)*(1.0-v)/4.5
                +   p/(5.0+15.0*(1.0-tanh(50.0*(u-0.85))) ) ;
   float  Isi   =   -w*(( 1.0-tanh(10.0*(u-0.9)) )/2.0)*
                   ( 1.0+tanh(7.*(u-0.35)))/
                   ((2.0*15.5)*(1.0+exp((u-0.9)*4.5))) ;
   if   ( u < 0.05 )   Isi = 0.0 ;

   du2dt -= (Ifi+Iso+Isi)/C_m ; /* adding reaction terms
                                   to du/dt            */
```

```
/*-------------------------------------------
 * The forward Euler time integration and updating variables
 *-------------------------------------------
 */
    C.r = u + du2dt*dt ;     /*    march u in red channel     */
    C.g = v + dv2dt*dt ;     /*    march v in green channel   */
    C.b = w + dw2dt*dt ;     /*    march w in blue channel    */

/*-------------------------------------------
 * ouputting the shader
 *-------------------------------------------
 */
  outUvw = C ; /* set the output as the updated color */
  return ;
}
```

Many lines of this code are self-explanatory, and comments have been added throughout to further clarify the purpose of each instruction. The variable declarations in the interface section of this code are the variables that arrive at the GPU, either from the CPU side or from previous GPU calculations. The interface variables are differentiated into three general categories: ins (also known as varyings), uniforms, and outs. Ins (varyings) are variables that can vary from pixel to pixel and are to be calculated in a separate part of the WebGL program called the vertex shader, which is mainly in charge of calculating the position of points and pixels in the graphical pipeline. All our computational codes use a generic vertex shader program, which can be seen below.

```
  precision highp float ;   /* high precision for floats   */
  out vec2 pixPos ;         /* position of the pixel center;
                               see the fragment shader for
                               more details.               */
  void main() {
    pixPos = position.xy ; /* set the pixel position to the
                              position of the points.      */
    /* The following line calculates
       the position of each pixel on the sreen. */
    gl_Position = vec4(position.x*2.-1., position.y*2.-1.,0.,1.0);
}
```

This code is identical in all our demonstrated cases and does not require modification. Uniforms are variables that are uniformly defined for all the pixels that are to be colored using the fragment shader. Outs are variables that are the output of the fragment shader for the particular pixel that is colored in the shader.

Because almost all the computation happens in the fragment shader, we concentrate on the fragment shader code. The most noteworthy variable declarations in the fragment code are the following:

```
  uniform sampler2D   inUvw ;
```

and

```
  layout (location = 0 ) out vec4 outUvw ;
```

where the former indicates a handle to the entire texture/image that enters the shader and the latter is the color calculated for the fragment/pixel through the shader. We should note that the input texture `inUvw`, which is of type sampler2D, is uniformly defined for all pixels. This implies that each pixel will have access to the entire texture/image for reading. However, each pixel writes its own value into the output texture.

The shader source codes must be passed to `Abubu.js` as string variables. Hence, the vertex and fragment shader source codes can be stored as JavaScript string variables, or they can be saved into text files separately and loaded at run time into JavaScript variables using an asynchronous JavaScript file loader such as require.js. Assuming that the source codes for the vertex and fragment shaders are already stored in JavaScript variables `compShader` and `vertShader`, we can define a program that receives the input textures and writes the output textures as follows.

```
var comp1 = new Abubu.Solver({
      vertexShader   : vertShader,
      fragmentShader : compShader,
      uniforms       : {
         inUvw    : { type : 't', value: fuvw  } ,
         ds_x     : { type : 'f', value: 8     } ,
         ds_y     : { type : 'f', value: 8     } ,
         dt       : { type : 'f', value: 0.02  } ,
         diffCoef : { type : 'f', value: 0.001 } ,
         C_m      : { type : 'f', value: 1.0   } ,
      } ,
      renderTargets: {
          outUvw : { location : 0 , target : suvw } ,
      }
}) ;
```

The above instruction automatically defines a WebGL program with the aforementioned source codes; automatically pairs the `fuvw` texture with `inUvw` in the shader source code; and sends the necessary values for the domain size, time-stepping information, etc., to the GPU. It also pairs the output of the program source code `outUvw` with the texture `suvw`. By using `Abubu.js`, this short snippet of code hides many details that otherwise would need to be implemented in a very peculiar way through numerous lines of code due to the internal complexities of the graphical pipeline. Whenever we are using one of the `Abubu.js` calls or calls to variables that have been defined using `Abubu.js`, the library hides various details of the WebGL setup and provides an abstracted environment that can be easily understood and implemented by a "novice" programmer. By calling the line

```
comp1.render() ;
```

the solution can be marched forward one time step in our JavaScript code from `fuvw` into `suvw`. To create a full time-stepping loop and to avoid swapping the textures without updating the solution once, we define a second solver with the same source code as follows.

```
var comp2 = new Abubu.Solver({
      vertexShader   :   vertShader,
      fragmentShader :   compShader,
      uniforms       :   {
         inUvw    : { type : 't', value: suvw  } ,
         ds_x     : { type : 'f', value: 8     } ,
         ds_y     : { type : 'f', value: 8     } ,
         dt       : { type : 'f', value: 0.02  } ,
```

```
        diffCoef: { type : 'f', value: 0.001} ,
        C_m     : { type : 'f', value: 1.0   } ,
    } ,
    renderTargets: {
        outUvw : { location : 0 , target : fuvw } ,
    }
}) ;
```

The only difference between `comp1` and `comp2` is that the pairing between input and output textures is swapped for `comp2`. This means that rendering `comp2` will result in marching the solution forward one time step from `suvw` into `fuvw`. Rendering `comp1` and `comp2` sequentially will result in updating the solution from `fuvw` into itself over two time steps without using the texture as both the input and the output simultaneously in any single time step update.

Additionally, we have implemented a few visualization tools in `Abubu.js` that can be easily incorporated in the code. `Plot2D` is one such tool. For example, by using the following block of code, we can set up a simple program to visualize the membrane potential as the computation progresses.

```
var  disp  =  new Abubu.Plot2D( {
       target    :   fuvw ,
       channel   :   'r',
       colormap  :   'jet',
       canvas    :   document.getElementById('canvas_1') ,
       minValue  :   0 ,
       maxValue  :   1.0 ,
} );
```

This code will create a colorplot of the red channel of the texture `fuvw` each time we call `disp.render();` in our JavaScript code. The canvas element, which actually displays the colorplot here, has the `id='canvas_1'` tag in the HTML code that is servicing the JavaScripts. It will use the "jet" colormap for colormapping. The range of values used for plotting will be between 0 and 1.

At this point, we can complete the time-marching and visualization loop of the program, which can be implemented as the following function.

```
var  run  =  function(){
     var  frameRate = 2400 ;     /* maximum number of time
                                    steps solved per second
                                    of wall-time, assuming
                                    that the screen refresh
                                    rate is 60 Hz.        */
     /* time-marching loop */
     for(var  i=0 ; i <frameRate/120 ; i++){
       comp1.render() ;
       comp2.render() ;
    }
    disp.render() ;
    requestAnimationFrame(run) ;
}
```

By calling this function once, we will run the time-marching loop 20 times to update the solution for 40 time steps before we update the display canvas that was set up earlier. When the drawing process on the canvas is finished, we request another animation frame by recursively calling the same function again, and the infinite loop continues.

The `for` loop in the function is used to update the display less frequently as most screens refresh at 60 Hz. Thus, if we were to update the display every time step, the plotting part would become the bottleneck of the program and we would be able to advance the solution only 120 times per second of wall time (60Hz×2 timesteps/`disp.render()` = 120 time steps per second). With this loop, we can overcome this issue and advance the solution much faster, in this case, 2400 time steps per second of wall time (60Hz×40 timesteps/`disp.render()` = 2400 time steps per second). Depending on the chosen `frameRate` that we choose and the graphics card that is used, these solutions can become significantly faster. For example, on a NVIDIA TITAN X (Pascal) graphics card, it is possible to run up to 38,000 time steps per second of wall time for this model on 512512 grid. This means that the simulation in 2D runs faster than real time; in particular, for a side-by-side view of an experiment of a 2D monolayer of porcine tissue (6 cm × 6 cm) and a simulation on the model on a TITAN X of the same size, the simulation would be at least three times faster.

Extension of WebGL computational codes to 3D settings

As mentioned earlier, the primary data structure for numerical computations in the library is a rectangular grid/image. This type of data structure increases the efficiency of the WebGL programs through simplifying the parallelization and workload balancing on the GPU. Although the underlying data structures in the WebGL applications remain rectangular grids/images, the extension to 3D simulations is still straightforward and can be achieved by considering the entire domain to be a large image that is a grid of sub-images, where each sub-image corresponds to a slice of the third dimension. Assuming that the data are arranged in an mx by my grid, we can use the following function in our fragment shaders to access the data structure.

```
// Accessing 3D coordinate (texCoord) of 'S' sampler
vec4 Texture3D( sampler2D S, vec3 texCoord )
{
   vec4    vColor1, vColor2 ;       /* colors on bottom and top
                                       slices */
   float   x, y ;                   /* coordinate on the 2D
                                       data structure */
   float   wd = mx*my - 1.0 ;       /* max slice number in S */
   float zSliceNo =
       floor( texCoord.z*mx*my) ;     /* bottom slice no    */

   x = texCoord.x / mx ;
   y = texCoord.y / my ;

   x += (mod(zSliceNo,mx)/mx) ;
   y += floor((wd-zSliceNo)/ mx )/my ;

   vColor1  =   texture( S, vec2(x,y) ) ;   /* color on bottom
                                               slice         */
   zSliceNo  =  ceil( texCoord.z*mx*my) ;  /* top slice no */

   x = texCoord.x / mx ;
   y = texCoord.y / my ;

   x += (mod(zSliceNo,mx)/mx) ;
   y += floor((wd-zSliceNo)/ mx )/my ;
   vColor2 = texture( S,  vec2(x,y) ) ; /* color on top slice */

   // Interpolating between the top and bottom slice to
   // get the color for the texCoord.z
   return  mix(
```

```
        vColor2,
        vColor1,
        zSliceNo/(mx*my)-texCoord.z
    );
}
```

Using such a data structure actually facilitates importing personalized data. As needed, computerized tomography (CT) scan data for the ventricular or atrial structures can be easily imported into the WebGL programs as images because segmentations will come exactly in that format and avoid the need of meshing or remeshing. Similarly, fiber orientation data can be imported and used in the WebGL programs in the same way from diffusion tensor MRI images, further increasing the facility of the programs of the library. In this work, the irregular boundaries from the 3D structures are handled by a well-established phase-field method (*30*). However, it is possible to use the same image data structures to store connectivity and coordinate information about computational meshes to be used for methods such as finite volume or finite elements.

Experimental methods

To develop some of the models used for this study as well as to compare and quantify the results obtained from these models when simulated in full 3D anatomically accurate ventricular structures to experiments in similar conditions, we have performed microelectrode and optical mapping experiments in rabbit and porcine hearts.

All animal experiments were approved by the Animal Care Committee of Georgia Tech, Atlanta, and were carried out in accordance with the *Guide for the Care and Use of Laboratory Animals*, published by the U.S. Public Health Service. Methods for obtaining the preparations are as follows. After anesthesia with Fatal-Plus (pentobarbital sodium, 390 mg ml^{-1}; Vortech Pharmaceuticals Ltd.; 86 mg kg^{-1}, intravenously), hearts were excised rapidly via a left thoracotomy and placed in cold, aerated (95% O_2–5% CO_2) Tyrode solution containing 124 mM NaCl, 4.0 mM KCl, 24 mM $NaHCO_3$, 0.9 mM NaH_2PO_4, 2.0 mM $CaCl_2$, 0.7 mM $MgCl_2$, and 5.5 mM glucose, adjusted to pH 7.4 with NaOH. For small hearts, the heart was cannulated from the aorta, and for larger hearts, the right and left coronary arteries were cannulated individually and the heart was perfused with Tyrode solution.

Microelectrode recordings

To fit the AP shape to the MM, microelectrodes were used on the surface of the heart. The preparations were stimulated using rectangular pulses of 2-ms duration and two to three times the diastolic threshold current (0.1 to 0.3 mA) delivered through Teflon-coated bipolar silver electrodes. Transmembrane APs at different pacing cycle lengths were recorded at 1 kHz using standard microelectrodes filled with 3 mM KCl. Examples of APs recorded at steady state in a variety of tissue preparations are shown in figs. S2 and S5. Such APs exhibit much less noise and a more pronounced upstroke than those obtained using optical mapping.

Optical mapping for voltage propagation in whole hearts

Electrical activity was assessed by voltage optical mapping using fluorescent imaging to map activation waves on the epicardial surfaces. Blebbistatin (10 μM) was added to the perfusate to prevent motion artifacts. Perfusion pressure was 50 to 80 mmHg, flow rate was 25 ml min^{-1}, and physiological temperature was maintained at 37.0° to 38.0°C. After equilibration, the preparation was stained with di-4-ANEPPS (10 μM), a voltage-sensitive dye.

For optical mapping, excitation light was produced by 18 high-performance light-emitting diodes (Luxeon III Star, LXHL-FM3C; wavelength, 530 ± 20 nm), 9 for the top view and 9 for the bottom view, driven by a low-noise constant current source. The illumination efficiency was enhanced significantly by collimator lenses (Luxeon, LXHL-NX05). The fluorescence emission light was collected by a Navitar lens (DO-2595; focal length, 25 mm; F/no., 0.95), passed through a long-pass filter (<610 nm), and imaged by 128128 back-illuminated electron-multiplied charge-coupled device array (Photometrics Cascade 128+), a camera providing high quantum efficiency (QE) (peak QE > 90%). The signal was digitized with a 16-bit A/D converter at a frame rate of 511 Hz (full frame, 128128 pixels). The peripheral component interconnect (PCI) interface provided high-bandwidth uninterrupted data transfer. An acquisition toolbox using C and Java was developed and used for experimental control, display, and data analysis, together with custom-made drivers for camera control and readout developed using C and OpenGL. Further description of protocols for measuring AP duration and conduction velocity restitution curves as well as for initiation and termination of fibrillation and for data analysis can be found in the Supplementary Materials.

RESULTS

Here, we show for the first time the feasibility of performing fast interactive simulations of large problems such as heart arrhythmias, which traditionally require supercomputers (*31*), locally on a PC. We illustrate some of the new possibilities that these fast simulations can provide along with direct quantification of 3D simulations with optical mapping experiments in full hearts. We also show the versatility of our approach using the new library by illustrating applications to other fields such as turbulent fluid flow and crystal growth.

Modeling 2D heart tissue

Many of the most dangerous and deadliest electrical arrhythmias such as tachycardia and fibrillation originate via reentrant waves (spiral waves) of electrical activity. Spiral waves can exhibit a large range of behavior from pinned spirals, attached to anatomical heterogeneities, to functional spirals whose dynamics is given by the electrophysiological conditions of the tissue. Spiral waves can thus be stable with a variety of tip trajectories (*32*) or can be unstable via many mechanisms (*33*). In Fig. 1 (A and B), we show two types of spiral wave reentry obtained experimentally from optical mapping: one in the high-excitable regime following a linear core trajectory and one in the lower excitable limit following a circular core. Below are voltage snapshots from corresponding simulations using the OVVR human cell model simulated in space with excitability parameters modified to reproduce the wavelength, period, and tip trajectory. Corresponding movies can be found in the Supplementary Materials. Other types of tip trajectories such as cycloidal, epicycloidal, hypocycloidal, and complex meandering along with different breakup mechanisms are shown in fig. S6 with links to interactive WebGL codes in Supplementary Programs.

It is important for patient-specific applications that models have correct behavior, and because tissue behavior is not always well predicted from cells alone (*14*, *34*), there is a need to perform parameter studies to validate models and their emergent tissue-level behavior. For example, there is no theory to predict, given a model in a single cell, whether spiral waves will follow linear or circular cores (Fig. 1, A and B)

or any transitional behavior in between (see the Supplementary Materials for details and supplementary WebGL applications for links to models).

While much effort has been made to account for the effects of model parameter values (*13*) and electrophysiological variability in cardiac cell models (*35*, *36*), no similar efforts exist to investigate the same phenomena in spatially extended systems, where the dynamics can be very different. Using a WebGL program, it is possible to quickly generate a study in space (2D and 3D) of wave stability and behavior as a function of key model parameters. For example, Fig. 1C shows for the first time a parameter space study for a complex cell model performed in tissue using the dynamics of the OVVR model in 2D as a function of the L-type calcium versus sodium channel conductances. The model, which consists of 41 differential equations per cell, was simulated for 10 s (to reach steady state) in an 8 cm × 8 cm domain (512 × 512 elements) for each of 36 different parameter combinations (27 shown in the figure); it took less than an hour to run all the parameter space simulations. A similar parameter space study for g_{to}, the fast transient outward potassium current conductance, versus a calcium activation time constant can be found in fig. S7. These two quick studies alone revealed a new type of dynamics not observed previously in cardiac models. The OVVR model has the potential for inducing early afterdepolarizations (EADs) from high-curvature regions that can destabilize spiral waves and lead to spiral wave breakup and chaos (fig. S7). EADs are irregular activations in voltage that can reactivate cardiac tissue and are thus considered proarrhythmic (*37*).

EAD behavior has been studied primarily in single cells, and much debate exists on the requirements needed for activations to overcome the small source-to-sink ratio for propagation (*38*). However, here for the OVVR model, several combinations of modifications of the sodium, calcium, and potassium current parameter values result in propagating EADs, most notable from highly concave regions of the wave back (Fig. 1C and fig. S7). Recently Kang *et al.* (*39*) studied the development of EADs using the OVVR model when

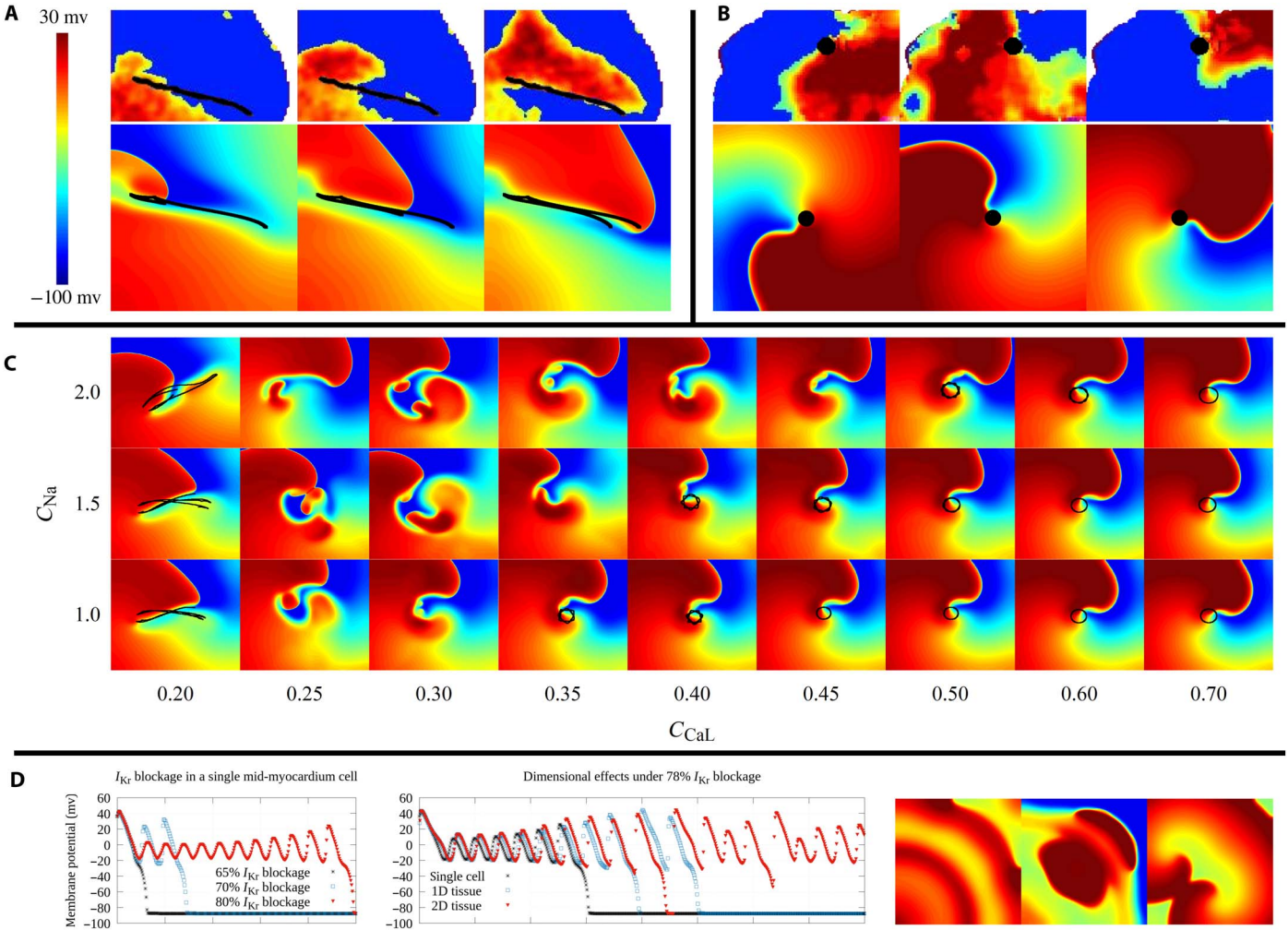

Fig. 1. Spiral waves of electrical activity in heart tissue. As indicated by the color bar, blue is tissue that is polarized at −80 mV and red is excited tissue that is depolarized at +20 mV. (**A** and **B**) Examples of spiral wave dynamics in two regimes. Experimental optical mapping (top) and numerical using the OVVR model (bottom) showing three snapshots from a rotating spiral waves that follows a linear core trajectory (left) and circular core trajectory (right) traced in black. (**C**) Example of parameter sweep using the OVVR model in 2D. A transition from linear to circular with some complex EAD formation in between is shown. (**D**) Effect of blocking the potassium I_{Kr} current and development of EADs that lead to fibrillation only in 2D. See the Supplementary Materials for examples of other tip trajectories, another parameter space study, and animations from the experimental data, and Supplementary Programs to run the WebGL codes.

blocking the rapid delayed rectifier potassium current (I_{Kr}), but only in an isolated cell due to the complexity of this model. Here, we extend their analysis from an isolated cell to 1D cables and to 2D tissues (Fig. 1D). Noteworthy finds are that when I_{Kr} is blocked by 78% and a single depolarizing stimulus is applied, no arrhythmic behavior appears in an isolated cell or in 1D, whereas in 2D tissue, sustained fibrillation results (see Fig. 1D, as well as movies in the Supplementary Materials and WebGL code in Supplementary Programs). This example of a single activation leading to chaotic behavior in 2D has not been shown before in any cardiac model. The short I_{Kr} blockage studies summarized in Fig. 1D took less than 7 min on an NVIDIA 1080Ti GPU to perform using WebGL for the epi, endo, and M cell versions of the OVVR model. It is also important to mention that only the M cell version of the model gave EADs; the epi and endo versions were not able to induce EADs in an isolated cell or in tissue.

Modeling 3D heart tissue

In 3D cardiac tissue, more new behavior emerges. For example, as spiral waves become scroll waves in 3D driven by vortex filaments, topological effects are now possible. As shown in Fig. 2, instabilities in the filament (*33*) can elongate, curve, and twist it enough to create new filaments by folding and by collisions with boundaries due to a negative tension instability (*33*), rendering the system chaotic in 3D while its 2D counterpart is completely stable. The complex intramural dynamics of 3D vortices during tachycardia and fibrillation then become very important when considering anatomically accurate heart structures (*40*).

Before patient-specific modeling can be achieved, quantitative validations of modeling in 3D with experiments are needed. Figure 3 (A and B) presents results from a model validation example using the porcine ventricles. The parameters of the MM were fitted to experimental data collected by pacing porcine ventricles at different rates to obtain characteristic curves that describe the adaptation of the tissue to pacing period. These curves in physiology are known as AP restitution curves and conduction velocity restitution curves (see the Supplementary Materials). Once the AP shape and adaptation curves (figs. S2 and S3) were fitted following standard methods (*41*), simulations were performed in WebGL in a 3D accurate porcine ventricular structure; links to the code are in Supplementary Programs. Depending on initial conditions and stimulation locations, we could obtain sustained ventricular tachycardia (VT) with a single spiral wave or VF with multiple waves. Figure 3A shows two snapshots during the rotation of a spiral wave from a simulation and from a whole-heart optical mapping experiment. Figure 3B shows one snapshot during fibrillation for the simulation and experiment.

The 3D simulations (Fig. 3, C and D) agree quantitatively with experimental data (*42*, *43*), including our own (see the Supplementary Materials for details), thereby validating the individualized model. Overall, the model is able to reproduce several key properties from experiments, not only the AP shape (fig. S2) but also values for APD_{max} = 255 ± 40 ms and dv/dt_{max} = 130 ± 10 V/s in experiments versus 264 ms and 130 V/s, respectively, in the model. The minimal diastolic interval (DI) and AP duration (APD) obtained from steady-state restitution curves in the experiments were 45 ± 5 ms and 95 ± 5 ms in our experiments [57 ± 6 ms and 107 ± 6 ms in Banville *et al.* (*42*) versus 50 and 100 ms in the model]. We observed restitution curve slope > 1 for DI < 90 ms in our experiments and model, similar to the 85 ± 5 ms observed by Banville *et al.* (*42*); although there is a region with slope greater than 1, the model is not able to produce alternans in tissue, matching our experiments and those of Banville *et al.* (*42*). For the simulations in 3D ventricles, we found that a single spiral wave has a period of 176 ms versus 184 ± 15 ms in the experiments, and a restitution curve obtained using all the points in the tissue cluster close to the steady-state APD restitution curve in a way similar to our experiments and to those of Lee *et al.* (*43*), as shown in fig. S4. For the

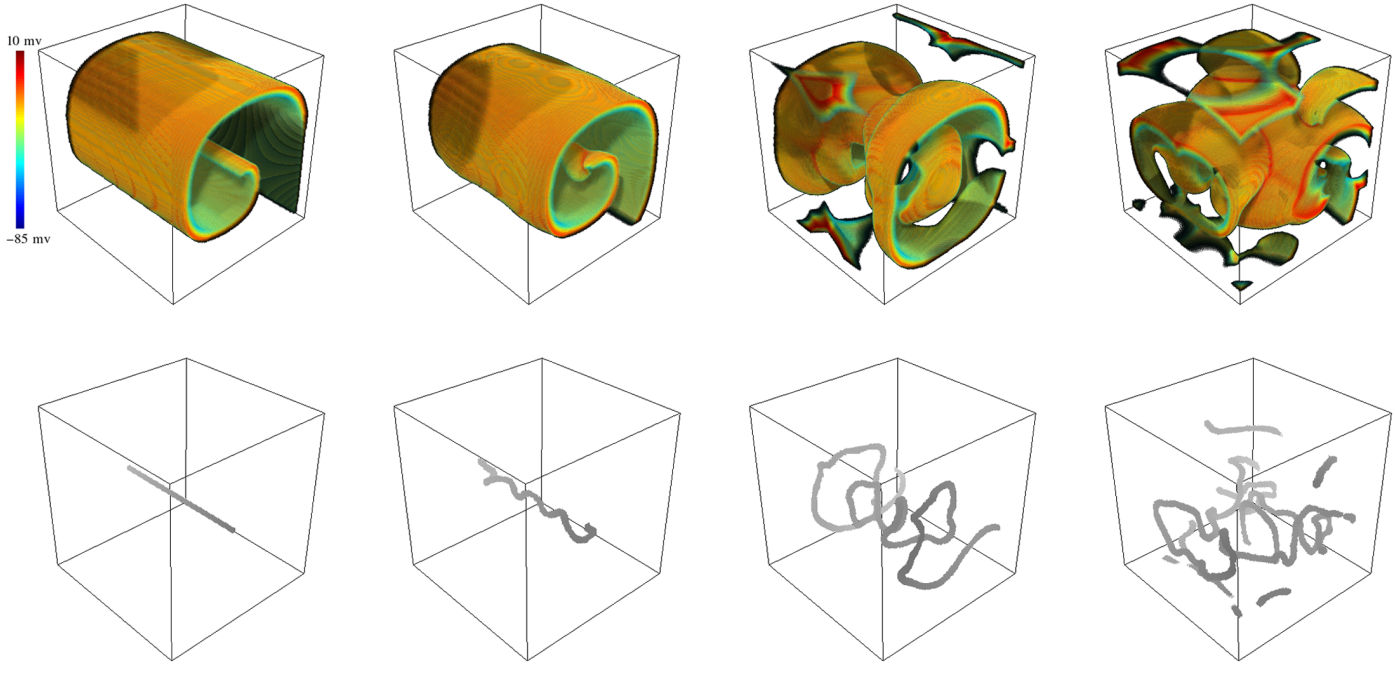

Fig. 2. Progression of an almost straight 3D scroll wave into complex turbulence in a box of 256 × 256 × 256 elements. Four snapshots of the evolution of a 3D scroll wave denoted by a voltage surface plot (**top**) and its vortex filament (**bottom**). A negative tension instability elongates the vortex filament that separates into new filaments as it touches the edges of the tissue. See Supplementary Programs to run the WebGL code.

case of sustained VF, the simulation shows a wider spread of the APD versus DI plot, as shown in fig. S4, indicating a lower dominant period as in our experiments and those of Banville *et al.* (*42*) and Lee *et al.* (*43*).

Another expected application of 3D heart modeling is testing the effectiveness of new pharmacological therapies for heart diseases along with verifying that other drugs are safe for the heart, as required by the FDA. Here, we present an example to simulate the effects of two drugs on rabbit ventricles. The MM was fitted to reproduce the effects of two drugs, diacetyl monoxime (DAM) and cytochalasin D (CytoD), after which validation was performed within 3D rabbit ventricles similarly to the porcine case already described. As shown in fig. S5, the MM was fitted to reproduce the rabbit AP shape and upstroke from microelectrode data for both DAM and CytoD, as well as their AP duration and conduction velocity restitution curves from optical mapping data. Simulations of these two drugs in 3D rabbit ventricles agreed with previous experiments (*44*) as well as our own experiments using optical mapping of stable tachycardia under DAM and fibrillation under CytoD with quantitatively matching dominant frequencies.

Figure 3 (C and D) shows the simulation and experimental (optical mapping) results for rabbit ventricles with DAM and CytoD. DAM decreases the APD and the ventricles can only sustain a single spiral wave, whereas for CytoD, the APD is longer and there is more interaction between the wave front and back when a spiral is initiated. This interaction can produce conduction block, leading to spiral wave breakup and fibrillation that turns out to be transient, as the multiple spiral waves are not able to fit in the tissue size. We found the spiral wave dominant period to be 134 ms in the 3D simulations versus 130 ± 10 ms in the experiments for DAM and a broader range of periods for CytoD with a peak centered at a higher period of 147 ms versus 150 ± 15 ms in the experiments. Similar behavior has been observed previously by Banville and Gray (*44*).

Personalized therapies will require running physiological models in human anatomies. We show an example of this by simulating arrhythmic conditions in a human atrial structure with the MM fitted to electrophysiological clinical data obtained from five different patients (*45*). The model was fitted to reproduce the APs obtained from a catheter at various frequencies of stimulation. Figure 3 (E and F) illustrates modeling results obtained using the models for the two different patients, showing one with a single stable spiral wave (flutter) and another with multiple waves (AF), as observed in the corresponding clinical cases. These simulations of the MM can be performed currently on a computer (with NVIDIA Titan-V GPU) at a speed that is 1/3 real time—that is, 1 s of arrhythmia in the patient is simulated in just 3 s. See Supplementary Programs for links to the WebGL models.

High-performance computing on cell phones

Because WebGL programs, by design, are independent of device and operating system, they can be executed on any relatively modern device with a GPU, from PCs to tablets and up to cell phones, with the main restriction simply the available memory. Some high-end phones have enough memory and powerful GPUs to run even 3D heart simulations interactively and, in some cases, faster than older PCs. Figure 4 shows examples of simulations of the complex TP and OVVR models on a Galaxy S8 cell phone in 2D sheets and in 3D ventricles; the latter involves solving up to 1.7 billion ODEs per second (see the Supplementary Materials for movies) on the cell phone's GPU.

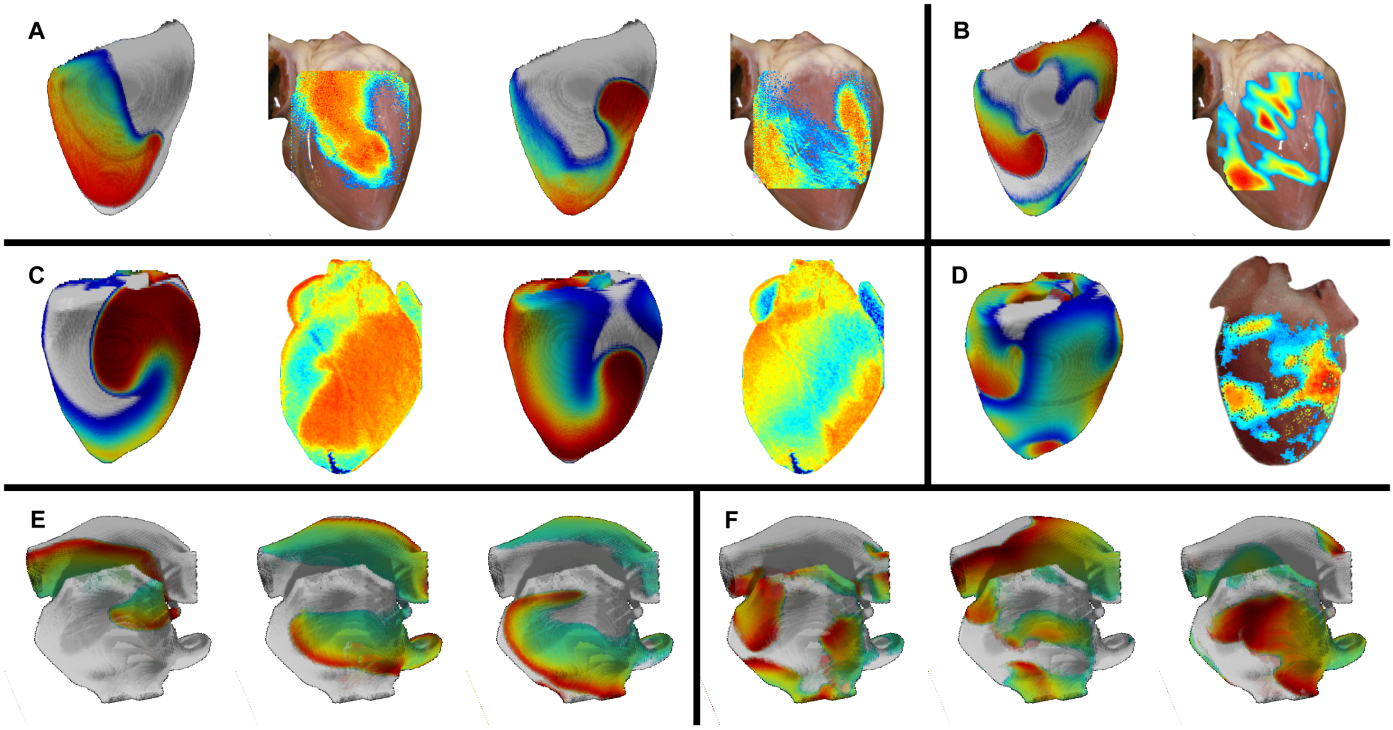

Fig. 3. Single and multiple spiral wave activity on realistic 3D heart geometries. (**A** to **D**) Comparing experimental data with the interactive simulations. (A) Single spiral wave (VT) and (B) fibrillation in porcine ventricles. (**C**) VT in rabbit with drug DAM. (D) Fibrillation in rabbit with drug CytoD. (**E** and **F**) Simulations of AF from models fitted to patient data. See the Supplementary Materials for animations and details and Supplementary Programs to run the WebGL codes.

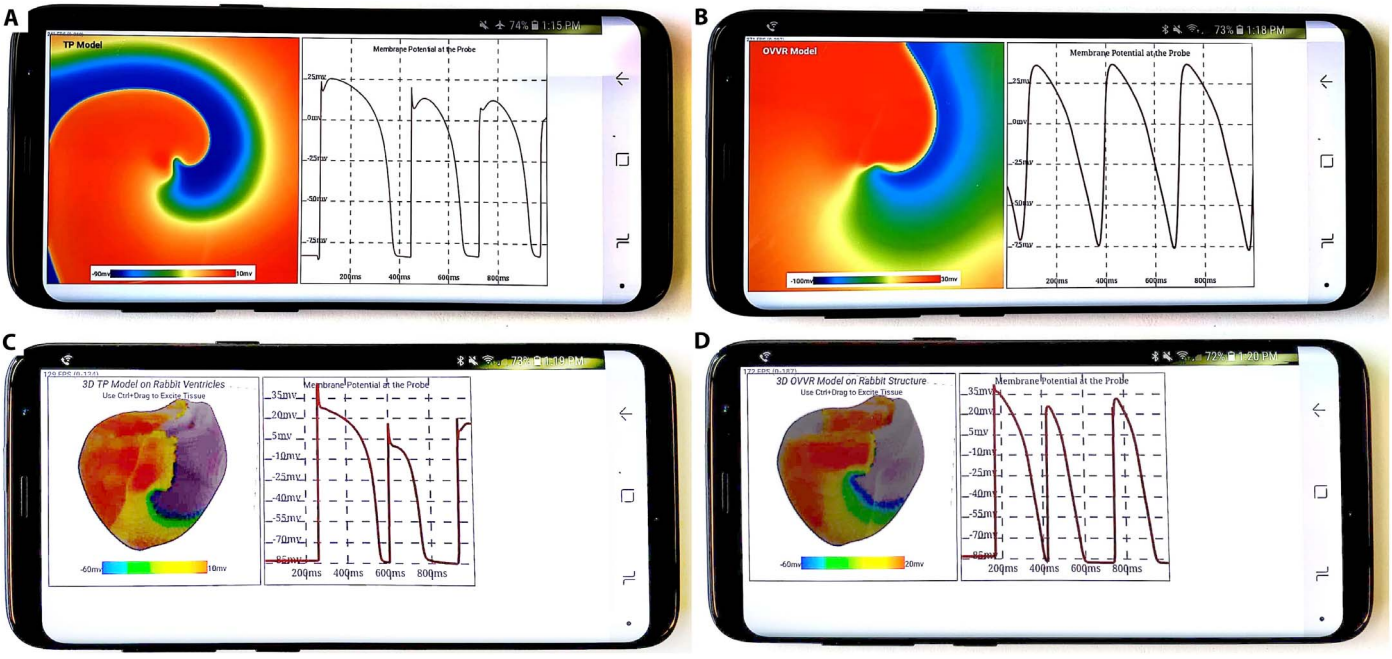

Fig. 4. High-performance simulations on a cell phone (Galaxy S8). (**A** and **B**) 2D spiral wave and (**C** and **D**) 3D reentry in rabbit ventricles with TP and OVVR models. Up to 1.7 billion ODEs can be solved per second using this phone. See the Supplementary Materials for details and animations and Supplementary Programs to run the WebGL codes.

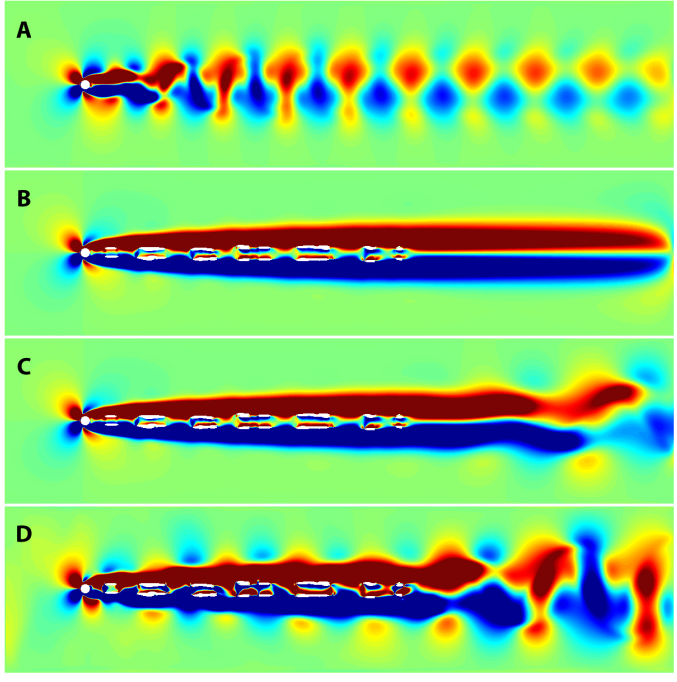

Fig. 5. High-performance simulations of fluid flow past a stationary cylinder in WebGL. (**A**) Vortex shedding from the stationary cylinder at *Re* = 54. (**B**) Vortex shedding is suppressed at *Re* = 54 by adding stationary obstacles downstream of the cylinder. (**C**) The Reynolds number is increased to *Re* = 64; we can see clear vortex shedding from the added obstacles, but the effects on the upstream cylinder are minimal. (**D**) Upon further increase of the Reynolds number to 80, the downstream vortex shedding becomes more pronounced, the flow around the upstream cylinder becomes unstable, and the oscillatory force is restored.

Applications to other fields

To present the capability of our library to implement large-scale noncardiac problems in WebGL, we have chosen to showcase two additional systems. First, we considered a fluid flow problem in the presence of obstacles implemented in WebGL using the `Abubu.js` library with a LBM (*46*). This WebGL application demonstrates how a series of quick simulations can help to predict fluid flow around obstacles at the design stage. Slip boundary conditions are applied at the top and bottom of the domain (*47*) for these simulations. A uniform velocity is applied at the inlet on the left side of the domain, and a stress-free boundary condition is applied at the right boundary of the domain. No-slip boundary conditions are used at the surface of the cylinder and any obstacle that is further added interactively at run time with the mouse. The domain size is 1200 × 300 lattice points. A classic D2Q9 lattice is used for the simulations, which implies that nine density equations need to be calculated at each time step for every lattice point. Using an NVIDIA Titan-V graphics card, we were able to solve 4500 time steps per second of wall time. The results are shown in Fig. 5, which shows the vorticity field. The obstacles, whose locations are fixed, are visualized in white. Figure 5A shows that the presence of a cylindrical object creates a clear Von-Karman vortex street for *Re* = 54. Figure 5B illustrates how it is possible to use these fast simulations to investigate how to suppress the vortex shedding by interactively adding obstacles in the downstream region of the cylinder. In Fig. 5C, we increased the Reynolds number to *Re* = 64. Trailing vortices began to form downstream from the cylinder; however, the disturbances were not strong enough to be felt at the cylinder location. Figure 5D shows that when the Reynolds number is increased to *Re* = 80, the disturbances in the flow become so pronounced that they can be felt even at the cylinder location. At this Reynolds number, the added downstream obstacles fail not only to suppress vortex shedding but also to shield the cylinder from the disturbances in the flow.

As a second example, we further implemented a crystallization problem in a supersaturated solution (*48*) in WebGL using `Abubu.js`. The WebGL implementation enabled us to perform some studies in a matter of seconds that usually would take much longer. Figure 6 shows the results of simulations in a grid size of 400 × 400 under different crystallization conditions. The lattice spacing, time step, saturation concentration, and viscosity of the fluid are all set to 1. The initial solute concentration is assumed to be 1.2 in all cases. In the top row of the figure, the Damkohler number is equal to 2, which indicates that the diffusion of the solute is the dominant process. In the bottom row, the Damkohler number is set to 160, which indicates a large rate of reaction (crystallization compared to diffusion). When diffusion is dominant, the crystal has a compact structure, as opposed to the case when the reaction is dominant and the crystal forms a branched structure. When there is no external fluid flow and a single nucleus is introduced at the center of the domain (Fig. 6, A and B), the crystals have a symmetrical structure. However, when fluid flow is introduced (here with a velocity of $u_x = 0.15$) while keeping the initial condition to a single nucleus at the center of the domain, the symmetry of the crystal is broken, as shown in Fig. 6 (C and D), and the crystal grows upstream where the higher concentrations of solute can replenish the crystallized solute. Figure 6C shows that with strong diffusion downstream of the flow, the solute cannot replenish as quickly as in the upstream section, and thick branches start to form downstream. Figure 6 (E and F) shows the effect of initial conditions with multiple nuclei when there is no external flow. In this scenario, when the diffusion is dominant (Fig. 6E), multiple nuclei can initiate crystallization, and the crystals can grow, merge, and form a large structure. However, when the reaction is the dominant process, multiple nuclei will initiate individual crystals that can grow, but the branches of crystals will not merge as crystallization depletes the solute and the solute cannot replenish fast enough.

DISCUSSION

Modeling can help in investigating the dynamics and effects of drugs in cardiac tissue (*9*, *11*, *12*). Here, we propose a significant step toward meeting the computational requirements for personalized computing-assisted treatment of cardiac electrophysiological diseases. We have developed a fast library (`Abubu.js`; see Supplementary Programs) that allows the solution of large, complex biophysical and physical systems interactively in near real time that until now could be solved only with supercomputers. We have shown an application of the library to cardiac dynamics along with other examples of complex systems such as fluid flow and crystal growth.

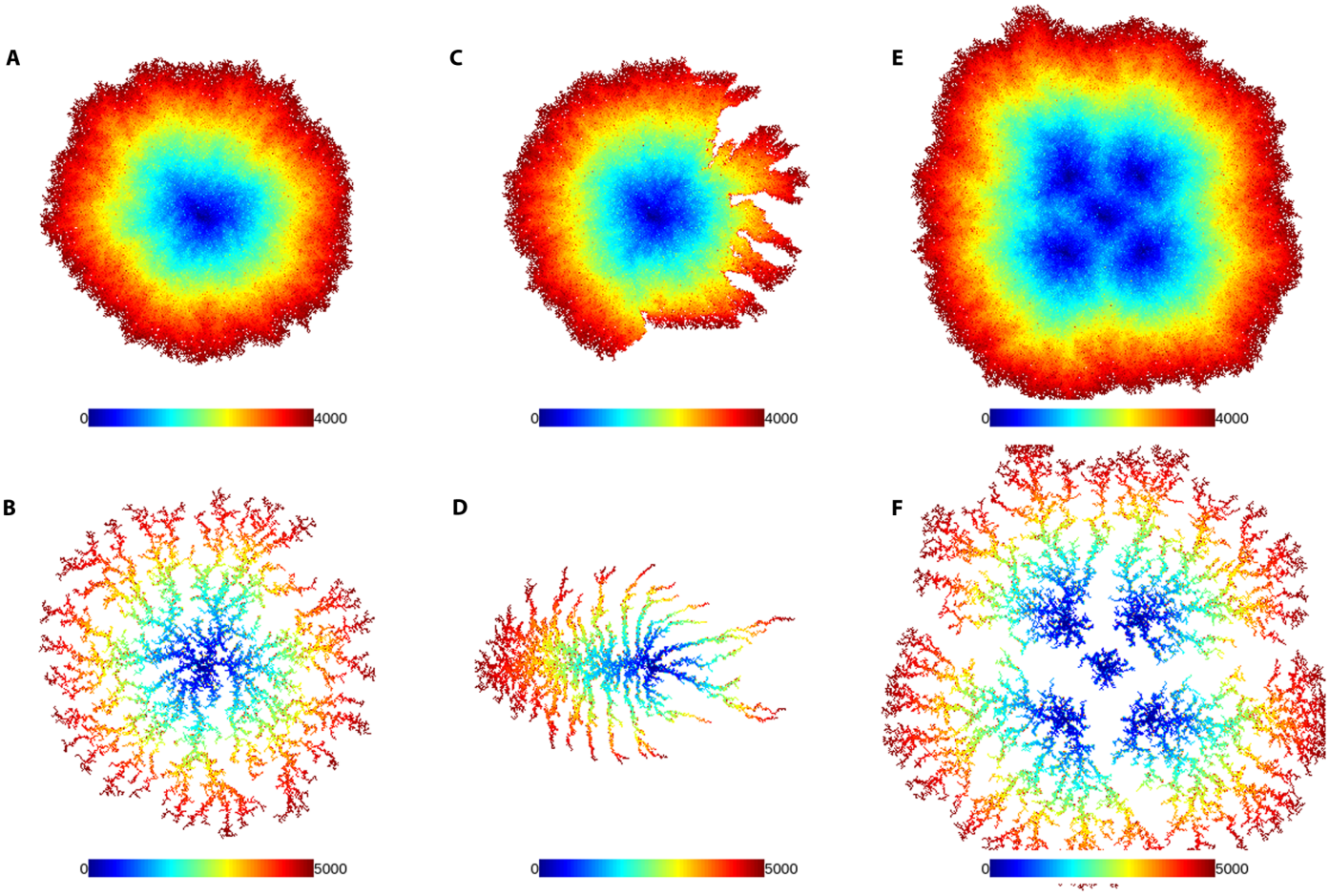

Fig. 6. High performance simulations of crystal growth under different Damkohler numbers, fluid flow conditions and number of nucleation cites using WebGL. Damkohler numbers are Da = 2 in (A), (C), and (E) and Da = 160 in (B), (D) and (F). Fluid flow conditions are $u_x = 0$ in (A), (B), (E) and (F) and $u_x = 0.15$ in (C) and (D). (**A**) to (**D**) have a single nucleation cite at the center of the domain, while (**E**) and (**F**) have five distinct nucleation sites. The crystals are colored on the basis of their crystallization time.

In the cardiac case, we show that it is possible to perform model studies as they are currently done in single cells (*11*–*13*), but now in space and in large tissue sizes, interactively, in near real time and by anyone who has a device with a medium- to high-end GPU currently in the range of U.S. $500 to 1000. This is the first time that complex cardiac models can be run easily and quickly on a PC or even a cell phone, allowing access to anyone interested in understanding how waves propagate in cardiac tissue in normal and arrhythmic cases. Therefore, now not only researchers but also doctors, patients, and the general public can actually use simulations to better explain and understand how different arrhythmias occur and how they could be treated and terminated, each group focusing on a different level of complexity as discussed in the next section.

For all of the 2D and 3D models presented here, the library simplifies interaction with the codes through the graphical user interface so that, for example, the tissue can be stimulated at any time using the mouse to produce activations that interact with existing waves and potentially can initiate or terminate an arrhythmia. Stimulation can also be induced and controlled in any part of the tissue by defining a period and stimulation size in the menu options to emulate pacing from an electrode. Users can investigate parameter effects dynamically and model, for example, how the enhancement or block of currents (as affected by a simulated drug) can actually change the dynamics of the waves in space in such a way as to increase or decrease arrhythmia incidence. Various menus of the programs allow not only modification of model parameter values interactively in real time but also visualization and plotting of different variables and currents; saving movies and data; and varying tissue sizes, resolution, and speed of simulations. The models used for all examples presented here and the library `Abubu.js` are freely available (in Supplementary Programs); users can modify them and add new physical, biological, or chemical models to investigate processes of interest interactively and in near real time.

Applications and limitations

The methodology and codes presented here allow fast interactive simulations of complex electrophysiological heart dynamics that until now were typically studied using parallel supercomputers and thus were restricted to only a handful of research groups around the world. By using the `Abubu.js` library, several of the complexities of writing GPU programs can be bypassed in favor of focusing just on implementing the models and numerical solution methods, thereby allowing more people to code and study cardiac arrhythmias as well as other large-scale systems like those we have shown here. Furthermore, several of the codes already developed and presented here have many options for changing model parameters and interactive options so that they can be of use in the study of arrhythmias by expert researchers and clinicians who are not expert programmers. Graduate, undergraduate, and even high school students can also use these programs to learn about the dynamics of complex systems, including pattern formation and chaos in biological, chemical, and physical systems. For example, we have successfully used some of these programs at national and international workshops including the 2017 and 2018 Undergraduate Workshop on Dynamics of Excitable Systems at Rochester Institute of Technology; the 2018 Hands-On Research in Complex Systems School at the International Center for Theoretical Physics in Trieste, Italy; a short course at the Federal University of Juiz de Fora, Brazil; and the 2018 Chaos High-School Summer Camp at Georgia Tech. In some cases, students not only have used the available codes to study complex dynamics but also have programmed models for Turing patterns consisting of two to six partial differential equations.

We have described a number of advantages of using WebGL and the `Abubu.js` library, which facilitate programming codes that run on a wider range of hardware. Although WebGL codes are by nature independent of operating system, some limitations still exist, often as a result of older hardware. For example, memory limitations on some devices may restrict the domain sizes that can be solved, thereby rendering some simulations impossible to run on that device. Also, older hardware, drivers, and operating systems may not support certain operations or certain types of data structures, so in some cases some of the codes that run well on some machines may have problems on others that have slightly older GPUs. However, we believe that support for different operations and data structures across different hardware and operating systems will continue to converge. For example, smart phones released more than two years ago do not support float textures and thus can only run WebGL codes that do not use these textures. However, most smart phones now support float textures and can even run simulations in 3D not possible a few years ago. Over time, we expect to see more powerful and affordable GPUs with better and homogeneous support on different devices. We have tested that all our programs presented here run on desktops under Windows and Linux with GPUs (from NVIDIA GTX-970 and up), a 2017 MacBook Pro (with Intel Iris Plus 460 GPU), a 2015 MacBook Pro (with AMD Radeon R9 M370X GPU), and Galaxy phones from S7 to S9.

Because the codes are running in the browser, some operating systems may use certain key combinations for shortcuts and define keys that exist only on a given system (e.g., Alt on PCs and option and command on Macs) or have touch pad control versus a mouse. While we have tried to make the programs run consistently across all platforms, there is still a possibility that the graphical interface, for example, may have small differences when using different systems and browsers, with Chrome and Firefox being the most homogeneous ones. Safari and Edge do not support WebGL 2.0 yet. Therefore, for applications that are to be deployed to end users, the general guidelines for code checking and compatibility must be followed by the developers who use the library. Instead of assuming the philosophy of "write once and run everywhere," we suggest following the general guideline of "write once, test everywhere, and then run everywhere." Finally, while the methods provided here can be used for any problem that can benefit from parallelism, problems that require very large memory, such as atmospheric simulations, cannot be tackled using our methods yet.

CONCLUSION

In this work, we present a methodology along with a library for accelerated interactive simulations of large-scale complex systems, which typically need high-performance supercomputer clusters, to speeds at or near real time on a common PC and even on cell phones. We show direct applications of this methodology to the simulation of large-scale cardiac modeling and describe several new contributions to cardiac dynamics while validating and quantifying these simulations. (i) We present an MM for porcine ventricular APs based on experimental data; to date, no mathematical model has been developed for porcine ventricular cells, so this represents the first model for porcine cardiac electrophysiology. We also present an MM for rabbit ventricular cells under the effects of two different drugs. The model parameters of these models were fitted to reproduce single-cell dynamics such as AP shape, threshold for excitation, rate of rise, and APD and CV restitution curves. (ii) These new models are simulated in space using our library and validated using experimental data not used in their development. The 3D simulations in ventricular

structures of porcine and rabbit ventricles quantitatively match the dynamics measured using optical mapping experiments performed in our laboratory and by other groups (*42*–*44*). In particular, we reproduced the experimentally observed spiral-wave dynamics (wavelength and frequencies) of tachycardia and fibrillation in the porcine ventricles as well as the stability of reentrant waves in rabbit hearts under DAM, leading to stable spiral waves, and under CytoD, leading to transient fibrillation with comparable frequencies. The simulations on the rabbit and porcine heart structures run at or near real time, so soon it will be possible to perform studies of closed-loop real-time feedback control between simulations and experiments. (iii) We demonstrate for the first time the feasibility of performing parameter sensitivity studies of a complex model (here the OVVR) in tissue without the need for supercomputers and in a matter of minutes. This is of particular importance as the Cardiac Safety Research Consortium (sponsored by the FDA) proposed a new CiPA in which drug assessment can include testing using numerical modeling. Furthermore, the OVVR model became the recommended ventricular myocyte model to be used as a test bed; however, only single-cell numerical experiments are required under this initiative because of the model's complexity. Many studies indicate that the dynamics of a single cell can be very different from the dynamics of tissue (*14*, *34*). Now, it is possible to use numerical experiments to study the effects of drugs on the dynamics of both single cells and tissue in a matter of minutes. We present here as an example an extension of a recent study of the effect of blocking the rapid delayed rectifier potassium current in single cells (*39*) to tissue and for all three types of ventricular cells (epi, endo, and mid myocardium). (iv) These simulations showed that only the mid-myocardium version of the OVVR model leads to EADs, which in 2D can actually initiate fibrillation from a single stimulus. This shows a new mechanism of single-site activation, leading to sustained arrhythmic behavior in space, which has not been observed or simulated before in other cardiac cell models.

Overall, we show that, using the proposed methodology, local desktops are able to solve up to 40 billion differential equations per second (wall time). These times currently translate to solving what the FDA considers to be the most up-to-date cardiac cell model at speeds that are only from 3 to 10 times slower than real time in 2D and 3D, respectively, with some simpler models actually running several times faster than real time. While detailed studies and applications to the clinic typically would be run on high-end desktops, we have shown that even cell phone GPUs are powerful enough to simulate billions of differential equations per second and to run simulations of these models in 2D and even 3D anatomies. Thus, our approach offers for the first time direct simple access to studies of complex physiological models in 2D and 3D anatomically accurate structures.

Furthermore, this methodology can be also used to accelerate simulations of similar spatially extended reaction-diffusion systems such as neuronal and brain dynamics, cancer and tumor growth, and the spread of infectious diseases. Finally, we have shown the versatility of our library in the aid of solving other types of large-scale complex problems that are not necessarily reaction diffusion systems with examples of fluid flow with obstacles and crystal growth at different dendrite regimes.

SUPPLEMENTARY MATERIALS

Supplementary material for this article is available at http://advances.sciencemag.org/cgi/content/full/5/3/eaav6019/DC1

Fig. S1. This colormap is used in all subsequent animations, unless a colormap is explicitly displayed in the animation.
Fig. S2. Parametrized MM of porcine ventricular cells (red) based on experimental microelectrode data (black).
Fig. S3. APD and CV restitution curves obtained from optical mapping in a Langendorff-perfused porcine heart (circles) and the fit by the MM (red line).
Fig. S4. APD restitution data obtained in the simulation from steady-state pacing shown in red and scatter plot of APD versus DI obtained from the whole tissue in the presence of a single spiral wave (VT, left) and during sustained fibrillation (VF, right).
Fig. S5. MM for rabbit with CytoD and DAM.
Fig. S6. Each panel shows a different scenario that was successfully modeled using WebGL.
Fig. S7. Parameter sweep using the OVVR model in conjunction with TP-I_{Na} current kinetics.
Movie S1. Linear spiral wave core trajectories seen in Fig. 1A obtained experimentally in canine right ventricle from optical mapping (top) and numerically reproduced by the OVVR model calculated using WebGL 2.0 (bottom) with m-cells and 90% of L-type calcium blockage.
Movie S2. Circular spiral wave core trajectory seen in Fig. 1B obtained experimentally in canine atria from optical mapping (top) and numerically reproduced by the OVVR model calculated using WebGL 2.0 (bottom) with m-cells.
Movie S3. EADs that are generated in a 2D slab of tissue seen in Fig. 1D from a single stimulus in the bottom left corner of the domain.
Movie S4. Evolution and breakup of a single scroll wave and its corresponding filament in a 3D slab.
Movie S5. Reproducing a single spiral wave in a porcine ventricular structure numerically (left) versus the optical mapping data obtained experimentally on a Langendorff-perfused porcine heart (right) seen in Fig. 3A.
Movie S6. Reproducing a single spiral wave on a rabbit ventricular structure numerically (left) versus the optical mapping data obtained experimentally on a Langendorff-perfused rabbit heart (right) seen in Fig. 3C.
Movie S7. Solving approximately 350 time steps of the Beeler-Reuter model (*49*) on a 512 × 512 grid.
Movie S8. Solving approximately 300 time steps of the TP 19-variable model on a 512 × 512 grid.
Movie S9. Solving approximately 300 time steps of the OVVR 35-variable model on a 256 × 256 grid.
Movie S10. Solving approximately 350 time steps of the Beeler-Reuter eight-variable model (*49*) on a 64 × 64 × 64 grid.
Movie S11. Solving approximately 70 time steps of the Beeler-Reuter eight-variable model (*49*) on a 128 × 128 × 128 grid.
Movie S12. Solving approximately 340 time steps of the TP 19-variable model on a 64 × 64 × 64 grid.
Movie S13. Solving approximately 170 time steps of the OVVR 35-variable model on a 64 × 64 × 64 grid.
Experimental measurements and fitting MMs to experimental data
Simulations of spiral waves with different trajectories
Further example of parameter space study using the OVVR model in 2D
Supplementary Programs. All the WebGL programs that were used in this article, as well as the copy of `Abubu.js` library, are included here.

REFERENCES AND NOTES

1. E. Benjamin, M. J. Blaha, S. E. Chiuve, M. Cushman, S. R. Das, R. Deo, S. D. de Ferranti, J. Floyd, M. Fornage, C. Gillespie, C. R. Isasi, M. C. Jiménez, L. C. Jordan, S. E. Judd, D. Lackland, J. H. Lichtman, L. Lisabeth, S. Liu, C. T. Longenecker, R. H. Mackey, K. Matsushita, D. Mozaffarian, M. E. Mussolino, K. Nasir, R. W. Neumar, L. Palaniappan, D. K. Pandey, R. R. Thiagarajan, M. J. Reeves, M. Ritchey, C. J. Rodriguez, G. A. Roth, W. D. Rosamond, C. Sasson, A. Towfighi, C. W. Tsao, M. B. Turner, S. S. Virani, J. H. Voeks, J. Z. Willey, J. T. Wilkins, J. H. Wu, H. M. Alger, S. S. Wong, P. Muntner; American Heart Association Statistics Committee and Stroke Statistic Subcommittee, Heart Disease and Stroke Statistics—2017 Update: A report from the American Heart Association. *Circulation* **135**, e146–e603 (2017).
2. A. T. Winfree, Electrical turbulence in three-dimensional heart muscle. *Science* **266**, 1003–1006 (1994).
3. R. A. Gray, J. Jalife, A. V. Panfilov, W. T. Baxter, C. Cabo, J. M. Davidenko, A. M. Pertsov, Mechanisms of cardiac fibrillation. *Science* **270**, 1222–1223 (1995).
4. R. A. Gray, A. M. Pertsov, J. Jalife, Spatial and temporal organization during cardiac fibrillation. *Nature* **392**, 75–78 (1998).
5. D. Scherr, P. Khairy, S. Miyazaki, V. Aurillac-Lavignolle, P. Pascale, S. B. Wilton, K. Ramoul, Y. Komatsu, L. Roten, A. Jadidi, N. Linton, M. Pedersen, M. Daly, M. O'Neill, S. Knecht, R. Weerasooriya, T. Rostock, M. Manninger, H. Cochet, A. J. Shah, S. Yeim, A. Denis, N. Derval, M. Hocini, F. Sacher, M. Haissaguerre, P. Jais, Five-year outcome of catheter ablation of persistent atrial fibrillation using termination of atrial fibrillation as a procedural endpoint. *Circulation* **8**, 18–24 (2015).
6. F. H. Fenton, E. M. Cherry, Models of cardiac cell. *Scholarpedia* **3**, 1868 (2008).

7. M. Fink, S. A. Niederer, E. M. Cherry, F. H. Fenton, J. T. Koivumäki, G. Seemann, R. Thul, H. Zhang, F. B. Sachse, D. Beard, E. J. Crampin, N. P. Smith, Cardiac cell modelling: Observations from the heart of the cardiac physiome project. *Prog. Biophys. Mol. Biol.* **104**, 2–21 (2011).
8. N. Trayanova, Defibrillation of the heart: Insights into mechanisms from modelling studies. *Exp. Physiol.* **91**, 323–337 (2006).
9. J. D. Moreno, Z. I. Zhu, P. C. Yang, J. R. Bankston, M. T. Jeng, C. Kang, L. Wang, J. D. Bayer, D. J. Christini, N. A. Trayanova, C. M. Ripplinger, R. S. Kass, C. E. Clancy, A computational model to predict the effects of class I anti-arrhythmic drugs on ventricular rhythms. *Sci. Transl. Med.* **3**, 98ra83 (2011).
10. S. Zahid, K. N. Whyte, E. L. Schwarz, R. C. Blake III, P. M. Boyle, J. Chrispin, A. Prakosa, E. G. Ipek, F. Pashakhanloo, H. R. Halperin, H. Calkins, R. D. Berger, S. Nazarian, N. A. Trayanova, Feasibility of using patient-specific models and the "minimum cut" algorithm to predict optimal ablation targets for left atrial flutter. *Heart Rhythm* **13**, 1687–1698 (2016).
11. S. Dutta, K. C. Chang, K. A. Beattie, J. Sheng, P. N. Tran, W. W. Wu, M. Wu, D. G. Strauss, T. Colatsky, Z. Li, Optimization of an in silico cardiac cell model for proarrhythmia risk assessment. *Front. Physiol.* **8**, 616 (2017).
12. I. Cavero, H. Holzgrefe, CiPA: Ongoing testing, future qualification procedures, and pending issues. *J. Pharmacol. Toxicol. Methods* **76**, 27–37 (2015).
13. E. A. Sobie, Parameter sensitivity analysis in electrophysiological models using multivariable regression. *Biophys. J.* **96**, 1264–1274 (2009).
14. R. Clayton, O. Bernus, E. M. Cherry, H. Dierckx, F. H. Fenton, L. Mirabella, A. V. Panfilov, F. B. Sachse, G. Seemann, H. Zhang, Models of cardiac tissue electrophysiology: Progress, challenges and open questions. *Prog. Biophys. Mol. Biol.* **104**, 22–48 (2011).
15. J. Bragard, A. Simic, J. Elorza, R. O. Grigoriev, E. M. Cherry, R. F. Gilmour, N. F. Otani, F. H. Fenton, Shock-induced termination of reentrant cardiac arrhythmias: Comparing monophasic and biphasic shock protocols. *Chaos* **23**, 043119 (2013).
16. A. Bueno-Orovio, E. M. Cherry, F. H. Fenton, Minimal model for human ventricular action potentials in tissue. *J. Theor. Biol.* **253**, 544–560 (2008).
17. K. H. ten Tusscher, A. V. Panfilov, Alternans and spiral breakup in a human ventricular tissue model. *Am. J. Physiol. Heart Circ. Physiol.* **291**, H1088–H1100 (2006).
18. T. O'Hara, L. Virág, A. Varró, Y. Rudy, Simulation of the undiseased human cardiac ventricular action potential: Model formulation and experimental validation. *PLOS Comput. Biol.* **7**, e1002061 (2011).
19. R. C. Mysa, A. Kaboudian, R. K. Jaiman, On the origin of wake-induced vibration in two tandem circular cylinders at low Reynolds number. *J. Fluids Struct.* **61**, 76–98 (2016).
20. Y. Ohya, T. Karasudani, A shrouded wind turbine generating high output power with wind-lens technology. *Energies* **3**, 634–649 (2010).
21. M. M. Zdravkovich, *Flow Around Circular Cylinders, Volume 2: Applications* (Oxford Univ. Press, 2003).
22. A. A. Mohamad, *Lattice Boltzmann Method: Fundamentals and Engineering Applications with Computer Codes* (Springer Science & Business Media, 2011).
23. F. H. Fenton, E. M. Cherry, H. M. Hastings, S. J. Evans, Real-time computer simulations of excitable media: JAVA as a scientific language and as a wrapper for C and FORTRAN programs. *Biosystems* **64**, 73–96 (2002).
24. D. Barkley (2002); https://homepages.warwick.ac.uk/~masax/.
25. W. Wang, L. Xu, J. Cavazos, H. H. Huang, M. Kay, Fast acceleration of 2D wave propagation simulations using modern computational accelerators. *PLOS ONE* **9**, e86484 (2014).
26. B. Gouvêa de Barros, R. Sachetto Oliveira, W. Meira Jr., M. Lobosco, R. Weber dos Santos, Simulations of complex and microscopic models of cardiac electrophysiology powered by multi-GPU platforms. *Comput. Math. Methods Med.* **2012**, 824569 (2012).
27. E. Bartocci, E. M. Cherry, J. Glimm, R. Grosu, S. A. Smolka, F. H. Fenton, Toward real-time simulation of cardiac dynamics, in *Proceedings of the 9th International Conference on Computational Methods in Systems Biology* (ACM, 2011), pp. 103–112.
28. J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, T. J. Purcell, A survey of general-purpose computation on graphics hardware, in *Computer Graphics Forum* (Wiley Online Library, 2007), vol. 26, pp. 80–113.
29. A. L. Hodgkin, A. F. Huxley, A quantitative description of membrane current and its application to conduction and excitation in nerve. *J. Physiol.* **117**, 500–544 (1952).
30. F. H. Fenton, E. M. Cherry, A. Karma, W.-J. Rappel, Modeling wave propagation in realistic heart geometries using the phase-field method. *Chaos* **15**, 013502 (2005).
31. D. F. Richards, J. N. Glosli, E. W. Draeger, A. A. Mirin, B. Chan, J. L. Fattebert, W. D. Krauss, T. Oppelstrup, C. J. Butler, J. A. Gunnels, V. Gurev, C. Kim, J. Magerlein, M. Reumann, H. F. Wen, J. J. Rice, Towards real-time simulation of cardiac electrophysiology in a human heart at high resolution. *Comput. Methods Biomech. Biomed. Engin.* **16**, 802–805 (2013).
32. A. T. Winfree, Varieties of spiral wave behavior: An experimentalist's approach to the theory of excitable media. *Chaos* **1**, 303 (1991).
33. F. H. Fenton, E. M. Cherry, H. M. Hastings, S. J. Evans, Multiple mechanisms of spiral wave breakup in a model of cardiac electrical activity. *Chaos* **12**, 852–892 (2002).
34. E. M. Cherry, F. H. Fenton, A tale of two dogs: Analyzing two models of canine ventricular electrophysiology. *Am. J. Physiol. Heart Circ. Physiol.* **292**, H43–H55 (2007).
35. A. X. Sarkar, D. J. Christini, E. A. Sobie, Exploiting mathematical models to illuminate electrophysiological variability between individuals. *J. Physiol.* **590**, 2555–2567 (2012).
36. O. J. Britton, A. Bueno-Orovio, K. Van Ammel, H. R. Lu, R. Towart, D. J. Gallacher, B. Rodriguez, Experimentally calibrated population of models predicts and explains intersubject variability in cardiac cellular electrophysiology. *Proc. Natl. Acad. Sci. U.S.A.* **110**, E2098–E2105 (2013).
37. J. N. Weiss, A. Garfinkel, H. S. Karagueuzian, P.-S. Chen, Z. Qu, Early afterdepolarizations and cardiac arrhythmias. *Heart Rhythm* **7**, 1891–1899 (2010).
38. Y. Xie, D. Sato, A. Garfinkel, Z. Qu, J. N. Weiss, So little source, so much sink: Requirements for afterdepolarizations to propagate in tissue. *Biophys. J.* **99**, 1408–1415 (2010).
39. C. Kang, A. Badiceanu, J. A. Brennan, C. Gloschat, Y. Qiao, N. A. Trayanova, I. R. Efimov, β-adrenergic stimulation augments transmural dispersion of repolarization via modulation of delayed rectifier currents I_{Ks} and I_{Kr} in the human ventricle. *Sci. Rep.* **7**, 15922 (2017).
40. F. Fenton, A. Karma, Fiber-rotation-induced vortex turbulence in thick myocardium. *Phys. Rev. Lett.* **81**, 481 (1998).
41. D. I. Cairns, F. H. Fenton, E. Cherry, Efficient parameterization of cardiac action potential models using a genetic algorithm. *Chaos* **27**, 093922 (2017).
42. I. Banville, N. Chattipakorn, R. A. Gray, Restitution dynamics during pacing and arrhythmias in isolated pig hearts. *J. Cardiovasc. Electrophysiol.* **15**, 455–463 (2004).
43. M.-H. Lee, Z. Qu, G. A. Fishbein, S. T. Lamp, E. H. Chang, T. Ohara, O. Voroshilovsky, J. R. Kil, A. R. Hamzei, N. C. Wang, S. F. Lin, J. N. Weiss, A. Garfinkel, H. S. Karagueuzian, P. S. Chen, Patterns of wave break during ventricular fibrillation in isolated swine right ventricle. *Am. J. Physiol. Heart Circ. Physiol.* **281**, H253–H265 (2001).
44. I. Banville, R. A. Gray, Effect of action potential duration and conduction velocity restitution and their spatial dispersion on alternans and the stability of arrhythmias. *J. Cardiovasc. Electrophysiol.* **13**, 1141–1149 (2002).
45. D. M. Lombardo, F. H. Fenton, S. M. Narayan, W.-J. Rappel, Comparison of detailed and simplified models of human atrial myocytes to recapitulate patient specific properties. *PLOS Comput. Biol.* **12**, e1005060 (2016).
46. Z. Guo, C. Zheng, B. Shi, Discrete lattice effects on the forcing term in the lattice Boltzmann method. *Phys. Rev. E* **65**, 046308 (2002).
47. K. Wang, Z. Chai, G. Hou, W. Chen, S. Xu, Slip boundary condition for lattice Boltzmann modeling of liquid flows. *Comput. Fluids* **161**, 60–73 (2018).
48. Q. Kang, D. Zhang, P. C. Lichtner, I. N. Tsimpanogiannis, Lattice Boltzmann model for crystal growth from supersaturated solution. *Geophys. Res. Lett.* **31**, L21604 (2004).
49. G. W. Beeler, H. Reuter, Reconstruction of the action potential of ventricular myocardial fibres. *J. Physiol.* **268**, 177–210 (1977).

Acknowledgments: We would like to thank School of Physics at Georgia Institute of Technology and School of Mathematical Sciences at Rochester Institute of Technology for providing a healthy environment for research and collaboration. We would like to further extend our gratitude to E. E. Konjkav for tireless and inspiring moral support from the conception to the finalization of this work. **Funding:** This work was supported, in part, by the NSF (grants CNS-1446312 to E.M.C. and CNS-1446675 to F.H.F. and A.K.) and by the NIH (grant 1R01HL143450-01 to F.H.F., E.M.C., and A.K.). F.H.F., E.M.C., and A.K. also collaborated while at Kavli Institute for Theoretical Physics (KITP), and thus, research was also supported, in part, by NSF grant PHY-1748958, NIH grant R25GM067110, and Gordon and Betty Moore Foundation grant 2919.01. **Author contributions:** A.K., E.M.C., and F.H.F. contributed to the design of the study, the software and model development, and the writing of the manuscript. A.K. contributed to the design and implementation of `Abubu.js` library. A.K. and F.H.F. performed the experiments. A.K., F.H.F., and E.M.C. analyzed data. **Competing interests:** The authors declare that they have no competing interests. **Data and materials availability:** All programs are made available in the Supplementary Materials, and experimental data are available by request to the corresponding authors.

Submitted 2 October 2018
Accepted 14 December 2018
Published 27 March 2019
10.1126/sciadv.aav6019

Citation: A. Kaboudian, E. M. Cherry, F. H. Fenton, Real-time interactive simulations of large-scale systems on personal computers and cell phones: Toward patient-specific heart modeling and other applications. *Sci. Adv.* **5**, eaav6019 (2019).

Real-time interactive simulations of large-scale systems on personal computers and cell phones: Toward patient-specific heart modeling and other applications

Abouzar Kaboudian, Elizabeth M. Cherry and Flavio H. Fenton

Sci Adv **5** (3), eaav6019.
DOI: 10.1126/sciadv.aav6019