Transitioning from Visual Setup to text-based VSim simulations

Tom Jenkins

Senior Research Scientist

Tech-X Corporation

A brief introduction to me…

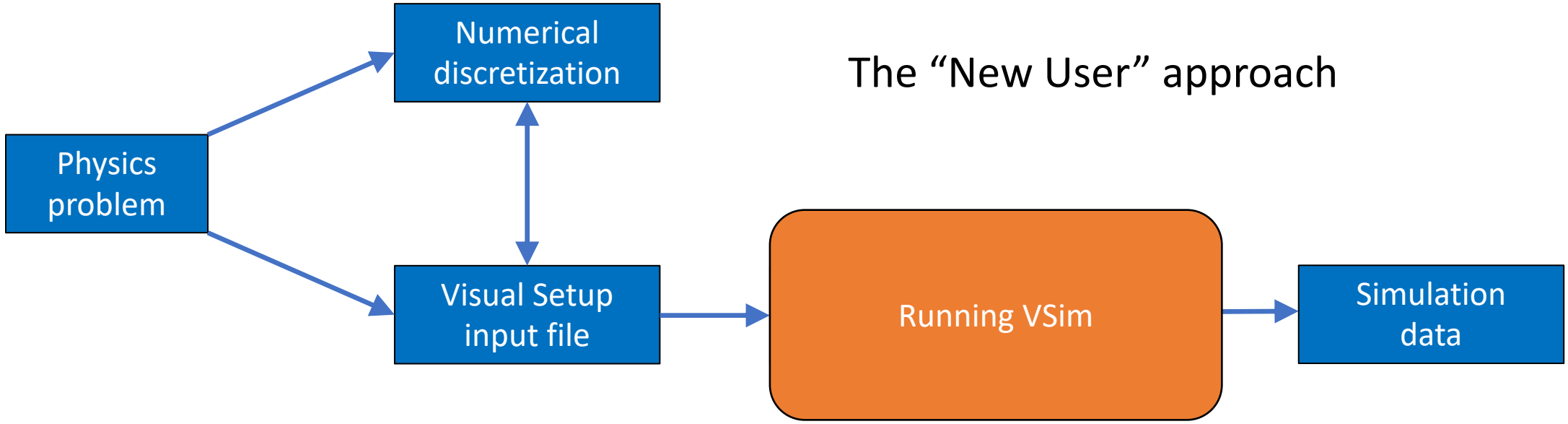
- Senior Research Scientist, 10.5 years at Tech-X
- Ph.D. @ Princeton/PPPL (2007), developing numerical methods for gyrokinetic PIC simulation
- Postdoc @ UW-Madison, working on RF/MHD coupling for electron cyclotron current drive in fusion plasmas
- Current research interests:
  - methods for speeding up particle-in-cell simulations (SLPIC)
  - modeling RF sheaths/impurity sputtering in fusion devices
  - kinetic theory – wave/particle interactions, etc.
  - PIC modeling of low-temperature plasmas
- Website, where this talk and many other talks/papers/presentations are posted:

  http://nucleus.txcorp.com/~tgjenkins

This talk focuses on how VSim works 'under the hood'

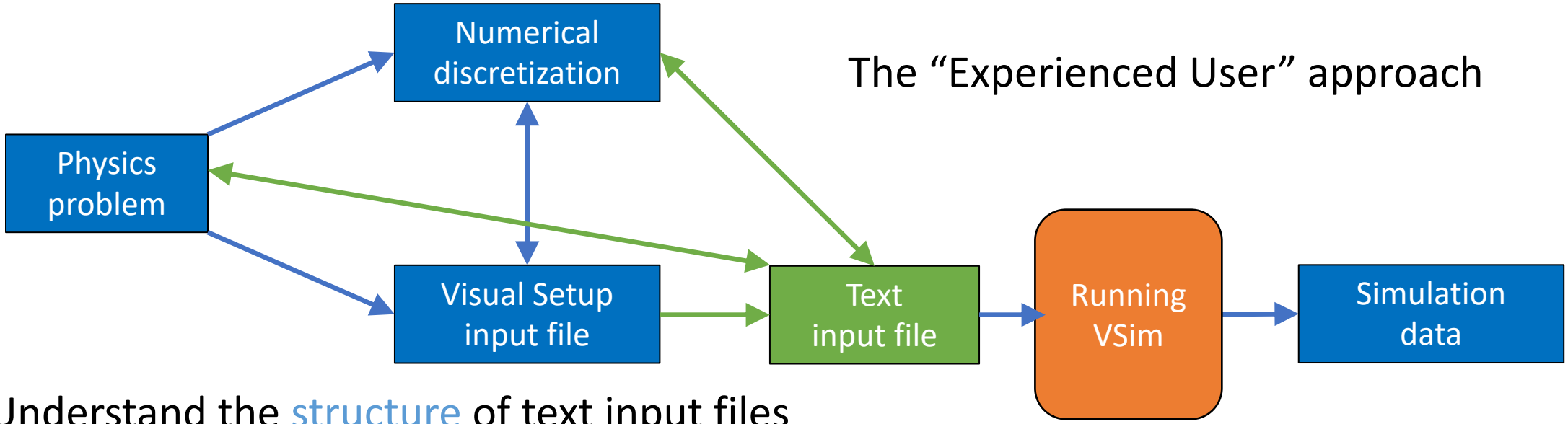
- VSim's Visual Setup interface is designed to quickly bring new users up the VSim learning curve
  - Allows common actions to be done quickly and systematically, with visual cues
    -defining grids -importing shapes
    -applying boundary conditions -adding particle species
  - Shows users options consistent with their previous choices, while hiding others
    -electrostatic vs. electromagnetic -direct vs. iterative matrix solve
- Visual Setup tools are adequate for many user needs (and we welcome suggestions for their improvement and development).
- Not everything that users want to do can be done in visual setup.
  - Exercise experimental or developing code features
  - Verify that the equations being solved are the ones the user intended
  - ?

My objective for this talk



The “New User” approach

My objective for this talk



The "Experienced User" approach

- Understand the structure of text input files
- Understand how numerical discretization techniques are implemented in text input files
- Understand how to add to/edit text input files to get the result you want
- Some complexity unavoidable! But we'll look at things in stages, and periodically review and regroup, to make things easier.

Useful resources for this talk

- VSim online documentation:
https://www.txcorp.com/images/docs/vsim/latest/VSimDocumentation.html

- Slides for this talk:
http://nucleus.txcorp.com/~tgjenkins/pres/TWSSTalk2020.pdf

- Download page for the VSim input files I will use in this talk:
http://nucleus.txcorp.com/~tgjenkins/TWSS2020.html

Choose a simple electrostatics problem: 1D Poisson

Physics problem

$$\frac{d^2\phi(x)}{dx^2} = -\frac{\rho(x)}{\epsilon_0} \quad ; \quad \phi(x=0) = \phi^{left}, \qquad \phi(x=L) = \phi^{right} \; ; x \in [0, L]$$

Numerical discretization

Numerical approach: discretize on a grid with *N* cells.

Define the grid: $\Delta x = \frac{L}{N} \quad ; \quad x_n = n\Delta x \quad \forall \; n = 0,1, \dots, N$

Use a finite-difference approximation to the second derivative, at interior gridpoints:

$$-\epsilon_0 \left[\frac{\phi_{j+1} - 2\phi_j + \phi_{j-1}}{\Delta x^2}\right] = \rho_j \qquad \forall \quad j = 1,2, \dots, N-1$$

Apply boundary conditions, at edge gridpoints:

$$\phi_0 = \phi^{left}$$
$$\phi_N = \phi^{right}$$

Solve the ensuing system of linear equations.

Solution error scales as $1/N^2$

Physics problem: exact solution

$$\frac{d^2\phi(x)}{dx^2} = -\frac{\rho_0 \sin\left(\frac{\pi x}{L}\right)}{\epsilon_0} \quad ; \quad \phi(x=0) = \phi^{left}, \qquad \phi(x=L) = \phi^{right} \quad on\ [0, L]$$

has exact solution

$$\phi(x) = \phi^{left} + \left(\phi^{right} - \phi^{left}\right)\frac{x}{L} + \frac{\rho_0 L^2}{\epsilon_0 \pi^2}\sin\left(\frac{\pi x}{L}\right)$$

Numerical discretization: approximate solution

On the discrete grid, we have

$$\phi_j^{exact} = \phi^{left} + \left(\phi^{right} - \phi^{left}\right)\frac{j}{N} + \frac{\rho_0 L^2}{\epsilon_0 \pi^2}\sin\left(\frac{\pi j}{N}\right) \quad ; \quad \rho_j^{exact} = \rho_0 \sin\left(\frac{\pi j}{N}\right)$$

Putting these functions into the discretized Poisson equation yields

$$-\frac{\rho_0}{\epsilon_0}\sin\left(\frac{\pi j}{N}\right)\left\{\frac{2N^2}{\pi^2}\left[1 - \cos\left(\frac{\pi}{N}\right)\right]\right\} \approx -\frac{\rho_0}{\epsilon_0}\sin\left(\frac{\pi j}{N}\right)$$

$$\left\{\frac{2N^2}{\pi^2}\left[1 - \left(1 - \frac{\pi^2}{2N^2} + \frac{\pi^4}{24N^4} + \cdots\right)\right]\right\} \approx 1$$

What does this look like in VSim?

Let's set up a basic simulation with Visual Setup and run it for one step:

Visual Setup input file

Parameters (5)
VLEFT = 0
VRIGHT = 1
LX = 1
NX = 10
RHOZERO = 20

Basic Settings (4)
number of steps = 1
steps between dumps = 1
dimensionality = 1
field solver = electrostatic

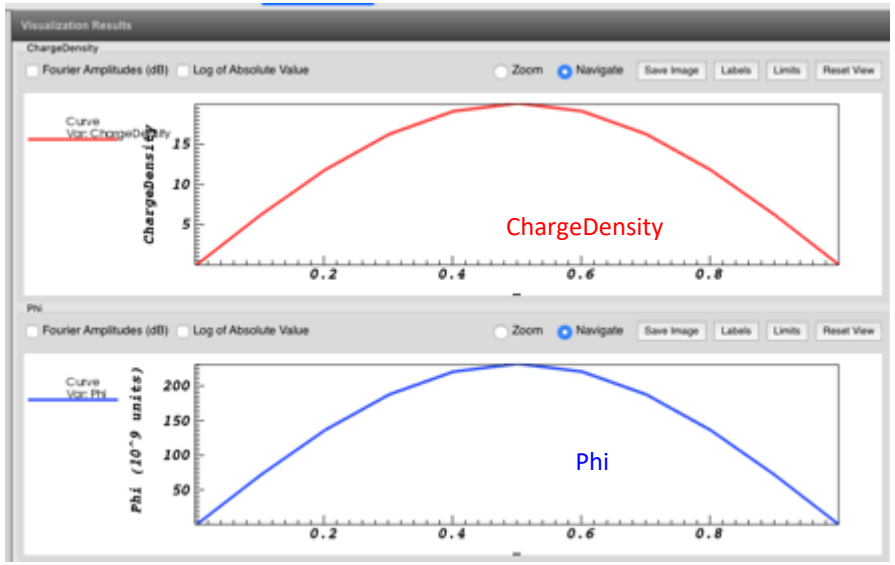
SpaceTimeFunctions (1)
RHOxt=RHOZERO*sin(PI*x/LX)

Grids (3)
xMin = 0
xMax = LX
xCells = NX

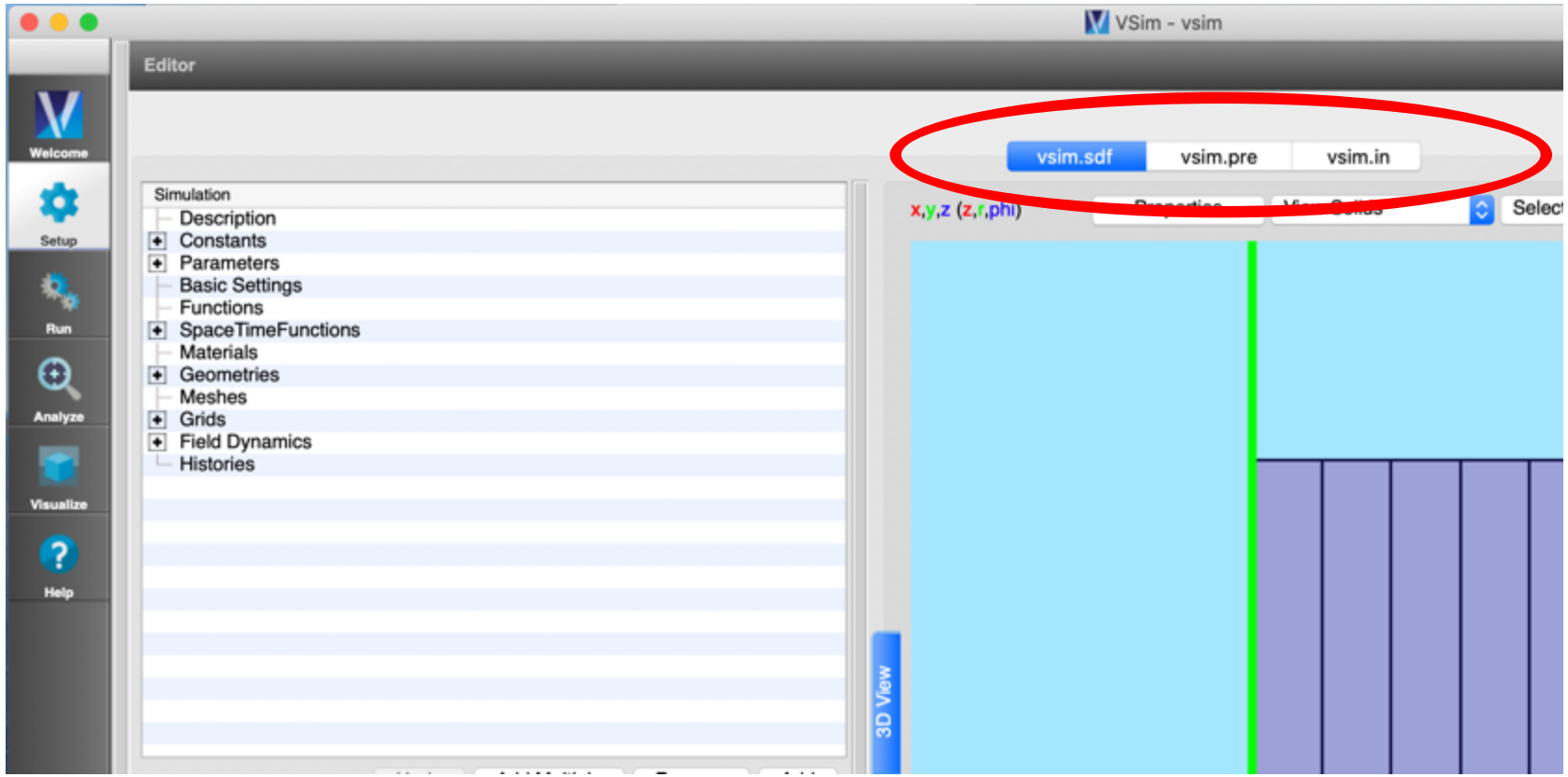
Field Dynamics: Fields (1)
Background Charge Density RHO=RHOxt

Field Dynamics: FieldBoundaryConditions (2)
Dirichlet on lower x: VLEFT
Dirichlet on upper x: VRIGHT

Field Dynamics: PoissonSolver (2)
preconditioner = no preconditioner
solver = SuperLU



VSim generates .pre and .in files from the Visual Setup .sdf file, when we Save and Setup



.pre file – an intermediate object not of much immediate use to us (if generated by Visual Setup from a .sdf file)

.in file – the text input file we want to learn how to work with

Looking at vsim.in – input blocks

```
Frontmatter

<Grid globalGrid>
  …
</Grid>

<Decomp decomp>
  …
</Decomp>

<MultiField NAME_OF_MULTIFIELD>

  <Field NAME_OF_FIELD>
    …
  </Field>

  <FieldUpdater NAME_OF_FIELDUPDATER>
    …
  </FieldUpdater>

  <InitialUpdateStep NAME_OF_INITIALUPDATESTEP>
    …
  </InitialUpdateStep>

  <UpdateStep NAME_OF_UPDATESTEP>
    …
  </UpdateStep>

  updateStepOrder = [NAME_OF_UPDATESTEP_1 NAME_OF_UPDATESTEP2 ...]
</MultiField>
```

Key VSim concept 0:
block structures

Or very generally,

```
<OBJECT objectName>
    …
  object features
    …
</OBJECT>
```

Looking at vsim.in – overall structure

```
Frontmatter

<Grid globalGrid>
  ...
</Grid>

<Decomp decomp>
  ...
</Decomp>

<MultiField NAME_OF_MULTIFIELD>

  <Field NAME_OF_FIELD>
    ...
  </Field>

  <FieldUpdater NAME_OF_FIELDUPDATER>
    ...
  </FieldUpdater>

  <InitialUpdateStep NAME_OF_INITIALUPDATESTEP>
    ...
  </InitialUpdateStep>

  <UpdateStep NAME_OF_UPDATESTEP>
    ...
  </UpdateStep>

  updateStepOrder = [NAME_OF_UPDATESTEP_1 NAME_OF_UPDATESTEP2 ...]
</MultiField>
```

Key VSim concept 1: the MultiField block

define *scalar or vector objects*: e.g. electric field, charge density, potential, …

define *mathematical operations* on field objects: e.g. taking the gradient of a scalar field and directing the output to a vector field

define *initial conditions* – done only once at simulation outset

Call the previously defined FieldUpdaters to manipulate the fields

in a specified *ordered sequence* of operations

Looking at vsim.in – Frontmatter, Grid block

defines some global simulation parameters

```
nsteps = 1
dumpPeriodicity = 1
dt = 1.0
dimension = 1
floattype = double
verbosity = 127
copyHistoryAtEachDump = 0
useGridBndryRestore = False
constructUniverse = False
```

number of steps in simulation

write data every 1 timestep

timestep (in units of seconds)

1D simulation

How detailed the VSim output should be ($= 2^M - 1$) ; larger M = more detail

defines spatial grid properties

```
<Grid globalGrid>
 verbosity = 127
 numCells = [10 11 12]
 lengths = [1.0 1.0 1.0]
 startPositions = [0.0 -0.5 0.0]
 maxCellXings = 1
</Grid>
```

3D grid: default y, z values

$\Delta x = 1/10$; $\Delta y = 1/11$; $\Delta z = 1/12$

(extra y, z dimensions are not used in this 1D computation, but may still be present in several parts of the input file)

Looking at vsim.in – Field blocks

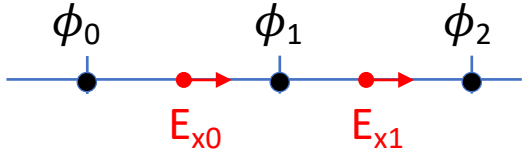
defines a scalar or vector field to be used in the simulation

```
<Field E>
  numComponents = 3
  offset = edge
  kind = regular
  overlap = [1 1]
  labels = [E_x  E_y  E_z]
</Field>
```

vector field

lives on edges between grid points

what to name the field components in the output file




```
<Field Phi>
  numComponents = 1
  offset = none
  kind = regular
  overlap = [1 2]
  labels = [Phi]
</Field>
```

scalar field

lives on grid points

default VSim field type

```
<Field ChargeDensity>
  numComponents = 1
  offset = none
  kind = depField
  overlap = [1 2]
  labels = [ChargeDensity]
</Field>
```

scalar field

lives on grid points

special VSim field type, built from particle data

messaging instructions for parallel computing:
include data from guard cells in a different way

Looking at vsim.in – FieldUpdater blocks

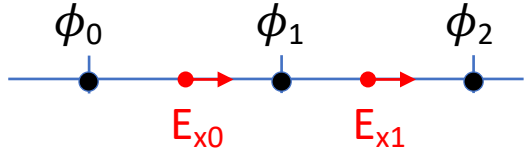
defines a mathematical operation on Field objects

built-in operation that computes the gradient of a scalar

```
<FieldUpdater gradPhi>
  kind = gradVecUpdater
  factor = -1.0
  lowerBounds = [0  0  0]          (inclusive)
  upperBounds = [10   11   12]     (exclusive)
  readFields = [Phi]
  writeFields = [E]
</FieldUpdater>
```

in other words,

$$\vec{E} = -\vec{\nabla}\phi.$$



names of previously defined Field blocks:
scalar input, vector output for this FieldUpdater kind.

```
<FieldUpdater RHO>
  kind = STFuncUpdater
  operation = add
  lowerBounds = [0  0  0]          (inclusive)
  upperBounds = [11  12  13]       (exclusive)
  writeFields = [ChargeDensity]
  component = 0
  cellsToUpdateAboveDomain = [False  False  False]
  <STFunc f>
    kind = expression
    expression = (20.0*sin(3.141592653589793*x/1.0))
  </STFunc>
</FieldUpdater>
```

built-in operation that manipulates SpaceTimeFunction objects

adds (subtracts, multiplies, etc.) the specified STFunc to the specified writeField

scalar

$= \rho_0 \sin\left(\frac{\pi x}{L}\right)$, from our input parameters

Looking at vsim.in – InitialUpdateStep blocks

Sets initial conditions for Field objects

These updates are performed only once, at the simulation outset.

```
<InitialUpdateStep RHOInitStep>
  alsoAfterRestore = True
  updaters = [RHO]
  messageFields = []
</InitialUpdateStep>

<InitialUpdateStep esSolveInitStep>
  alsoAfterRestore = True
  updaters = [esSolve]
  messageFields = [Phi]
</InitialUpdateStep>

<InitialUpdateStep gradPhiInitStep>
  alsoAfterRestore = True
  updaters = [gradPhi]
  messageFields = [E]
</InitialUpdateStep>
```

Also do this step when restarting a simulation

Previously defined field updater, defines ChargeDensity field

Previously defined field updater, solves Poisson equation for phi field

Previously defined field updater, computes E from phi.

Looking at vsim.in – UpdateStep blocks

These updates are performed at every timestep in the simulation.

Apply various FieldUpdater operations to Field objects, in a given sequence

```
<UpdateStep RHOStep>
  toDtFrac = 1.0
  updaters = [RHO]
  messageFields = []
</UpdateStep>

<UpdateStep esSolveStep>
  toDtFrac = 1.0
  updaters = [esSolve]
  messageFields = [Phi]
</UpdateStep>

<UpdateStep gradPhiStep>
  toDtFrac = 1.0
  updaters = [gradPhi]
  messageFields = [E]
</UpdateStep>
...
  updateStepOrder = [RHOStep esSolveStep gradPhiStep]
```

Advance the specified field to the next full timestep in this update

Previously defined field updater, defines ChargeDensity field (just as in InitialUpdateStep call)

Previously defined field updater, solves Poisson equation for phi field (just as in InitialUpdateStep call)

Previously defined field updater, computes E from phi (just as in InitialUpdateStep call).

UpdateSteps can appear in the input file in any order you like, the updateStepOrder determines which ones will be called when.

Regroup and Review

So far, we have:

-built an .sdf file in VSim, using Visual Setup, that solves the 1D Poisson equation

-found the .in text input file that VSim built from our initial .sdf file

-looked at the general block structure of that .in file

-looked at some typical blocks that live in the larger MultiField block, and their contents

*Field	*FieldUpdater
*InitialUpdateStep	*UpdateStep

Now, we'll do a bit of a deeper dive into how VSim solves the Poisson equation, and learn a bit more about how data is organized 'under the hood' in VSim.

Electrostatic solves, without VSim

Physics problem

VSim solves the Poisson equation

$$\frac{d^2\phi(x)}{dx^2} = -\frac{\rho(x)}{\epsilon_0} \quad ; \quad \phi(x=0) = \phi^{left}, \qquad \phi(x=L) = \phi^{right} \; ; x \in [0, L]$$

with Fields and FieldUpdaters and UpdateSteps.

Numerical discretization

Let's build a discretized version of this problem "by hand", to see what kinds of things we might expect VSim to be doing:

N-cell grid:

$$\Delta x = \frac{L}{N} \quad ; \quad x_n = n\Delta x \quad \forall \; n = 0,1,\dots,N$$

Discrete Poisson equation:

$$-\epsilon_0 \left[\frac{\phi_{j+1} - 2\phi_j + \phi_{j-1}}{\Delta x^2}\right] = \rho_j \qquad \forall \quad j = 1,2,\dots,N-1$$

Boundary conditions:

$$\phi_0 = \phi^{left}$$
$$\phi_N = \phi^{right}$$

Result: a linear system of equations for the unknown ϕ_j values.

Constructing the matrix – interior points

$$-\epsilon_0\left[\frac{\phi_{j+1}-2\phi_j+\phi_{j-1}}{\Delta x^2}\right]=\rho_j \qquad \forall \quad j=1,2,\ldots,N-1$$

becomes a matrix of form

$$\left(-\frac{\epsilon_0}{\Delta x^2}\right)\begin{bmatrix} 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 \end{bmatrix}\begin{bmatrix} \phi_0 \\ \phi_1 \\ \phi_2 \\ \vdots \\ \phi_{j-1} \\ \phi_j \\ \phi_{j+1} \\ \vdots \\ \phi_{N-2} \\ \phi_{N-1} \\ \phi_N \end{bmatrix}=\begin{bmatrix} \rho_0 \\ \rho_1 \\ \rho_2 \\ \vdots \\ \rho_{j-1} \\ \rho_j \\ \rho_{j+1} \\ \vdots \\ \rho_{N-2} \\ \rho_{N-1} \\ \rho_N \end{bmatrix}$$

This doesn't work for the first/last rows of matrix. Instead, we must use boundary conditions there.

Constructing the matrix – boundary conditions

$$\phi_0 = \phi^{left}$$
$$\phi_N = \phi^{right}$$

becomes

$$\left(\frac{-\epsilon_0}{\Delta x^2}\right)\begin{bmatrix} -\gamma\Delta x^2/\epsilon_0 & 0 & 0 & 0 & 0 & 0 & & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 & 0 & & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 & 0 & & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & 0 & 1 & -2 & 1 & 0 & & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -2 & 1 & & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -2 & & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 & \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 & \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\mu\Delta x^2/\epsilon_0 & \end{bmatrix}\begin{bmatrix} \phi_0 \\ \phi_1 \\ \phi_2 \\ \vdots \\ \phi_{j-1} \\ \phi_j \\ \phi_{j+1} \\ \vdots \\ \phi_{N-2} \\ \phi_{N-1} \\ \phi_N \end{bmatrix} = \begin{bmatrix} \gamma\phi^{left} \\ \rho_1 \\ \rho_2 \\ \vdots \\ \rho_{j-1} \\ \rho_j \\ \rho_{j+1} \\ \vdots \\ \rho_{N-2} \\ \rho_{N-1} \\ \mu\phi^{right} \end{bmatrix}$$

Changes in the right-hand side vector (charge density) are necessary to implement the BCs.

Rescaling factors γ, μ can be used to adjust the matrix condition number.

Canonical form: Ax = b.

linearSolveUpdater – solving the Poisson equation

Now let's look at how this is done in the vsim.in file.

One of VSim's built-in FieldUpdater blocks is the linearSolveUpdater, which solves equations of the form Ax = b.

Looking at vsim.in – linearSolveUpdater

A FieldUpdater object (mathematical operation) that solves a matrix equation Ax=b.

```
<FieldUpdater esSolve>
 kind = linearSolveUpdater
 lowerBounds = [0]   (inclusive)
 upperBounds = [11]  (exclusive)
 readFields = [ChargeDensity]
 readComponents = [0]
 writeFields = [Phi]
 writeComponents = [0]
 writeEquationToFile = 0

 <MatrixFiller interiorFiller>
  kind = stFuncStencilFiller
  verbosity = 127
  minDim = 1
  lowerBounds = [1 1 1]      (inclusive)
  upperBounds = [10 11 12]   (exclusive)
  component = 0

 <STFunc coeff>
  kind = expression
  expression = -8.854187817591624e-12
 </STFunc>
...
```

1D linear solve

Input: scalar ρ

Output: scalar ϕ

Can use this to look at the matrix (we will do this in a moment)

3D matrix template (even though we only need 1D)

$= -\epsilon_0$

MatrixFiller blocks do just what they sound like – filling rows in the matrix.

linearSolveUpdater - StencilElements

Inside the MatrixFiller block, we have various StencilElements:

```
<STFuncStencilElement phi_dxp>
 value = -100.0
 minDim = 1
 cellOffset = [0 0 0]
 functionOffset = [0.5 0. 0.]
 rowFieldIndex = 0
 columnFieldIndex = 0
</STFuncStencilElement>
```

No offset

$-1/\Delta x^2$

```
<STFuncStencilElement phi_npx>
 value = 100.0
 minDim = 1
 cellOffset = [1 0 0]
 functionOffset = [0.5 0. 0.]
 rowFieldIndex = 0
 columnFieldIndex = 0
</STFuncStencilElement>
```

+1 cell

$1/\Delta x^2$

functionOffset is irrelevant for node-centered fields

```
<STFuncStencilElement phi_dxm>
 value = -100.0
 minDim = 1
 cellOffset = [0 0 0]
 functionOffset = [-0.5 0. 0.]
 rowFieldIndex = 0
 columnFieldIndex = 0
</STFuncStencilElement>
```

No offset

```
<STFuncStencilElement phi_nmx>
 value = 100.0
 minDim = 1
 cellOffset = [-1 0 0]
 functionOffset = [-0.5 0. 0.]
 rowFieldIndex = 0
 columnFieldIndex = 0
</STFuncStencilElement>
```

-1 cell

n, d = non-diagonal or diagonal matrix element
m, p = - / +
cell/function offset

A generic interior row in the 1D Poisson matrix is

$$coefficient \cdot [\cdots \quad 0 \quad phi_{nmx} \quad (phi_{dxm}+phi_{dxp}) \quad phi_{npx} \quad 0 \quad \cdots]$$

linearSolveUpdater – boundary conditions

LHS (matrix)

```
<MatrixFiller RIGHTBCFiller>
 kind = stencilFiller
 verbosity = 127
 minDim = 1
 lowerBounds = [10 0 0]
 upperBounds = [11 12 13]
 component = 0

 <StencilElement ident>
  value = 1.7708375635183248e-09
  minDim = 0
  cellOffset = [0 0 0]
  rowFieldIndex = 0
  columnFieldIndex = 0
 </StencilElement>
</MatrixFiller>
```

Only rightmost cell

$= 2\epsilon_0/\Delta x^2$ (this is the μ factor from the earlier slide, on the LHS)

RHS (vector)

```
<VectorWriter RIGHTBCWriter>
 kind = stFuncVectorWriter
 verbosity = 127
 minDim = 1
 lowerBounds = [10 0 0]
 upperBounds = [11 12 13]
 component = 0

 <STFunc function>
  kind = expression
  expression = 1.0
 </STFunc>

scaling = 1.7708375635183248e-09
</VectorWriter>
```

Only rightmost cell

VRIGHT (chosen boundary condition)

$= 2\epsilon_0/\Delta x^2$ (again, the μ factor from the earlier slide, on the RHS)

linearSolveUpdater – the linearSolver block

```
<LinearSolver linearSolver>
 kind = directSolver
 solverType = superLU
 verbosity = 127
</LinearSolver>
```

Solve Ax = b by computing A^{-1} directly. Simplest VSim solver option (by the length-of-input-file metric, at least), but not useful if your matrix is too large.

All other VSim solver types are iterative:

- generalized minimum residual
- conjugate gradient
- biconjugate gradient
- etc.

Iterative solvers can be sped up by appropriate multigrid preconditioners (for which many options are available in VSim).

Let's look at the matrix VSim creates

- Edit the vsim.in file so that writeEquationToFile = 1.
- NOTE: If you now hit the "Save" button, VSim Composer will
  - re-read the vsim.sdf file, and
  - generate a new .in file from the information it finds there.
- This will overwrite the change you just made, since the sdf file defaults to writeEquationToFile = 0.
- Therefore: if you want to do text-based problem setup starting from a Visual Setup file, you'll need to generally do something like the following:
  - Generate the initial .in file from the sdf file with the "Save" button
  - Using your computer's file management utilities, copy the .in file to a .pre file with a different prefix name, e.g. vsim.in becomes vsimTextBased.pre
  - Edit this new .pre file in the way you want to
  - Open the modified .pre file in VSim, and run VSim as normal (the visual setup utilities will no longer work, but the physics engine will still parse and run the input file that you've modified)

Assuming Ax=b, A is in esSolveMatrix.mtx

```
%%MatrixMarket matrix coordinate real general
11 11 29
1 1 1.7708375635183248e-09
2 1 -8.8541900000000002e-10
2 2 1.7708380000000000e-09
2 3 -8.8541900000000002e-10
3 2 -8.8541900000000002e-10
3 3 1.7708380000000000e-09
3 4 -8.8541900000000002e-10
4 3 -8.8541900000000002e-10
4 4 1.7708380000000000e-09
4 5 -8.8541900000000002e-10
5 4 -8.8541900000000002e-10
5 5 1.7708380000000000e-09
5 6 -8.8541900000000002e-10
6 5 -8.8541900000000002e-10
6 6 1.7708380000000000e-09
6 7 -8.8541900000000002e-10
7 6 -8.8541900000000002e-10
7 7 1.7708380000000000e-09
7 8 -8.8541900000000002e-10
8 7 -8.8541900000000002e-10
8 8 1.7708380000000000e-09
8 9 -8.8541900000000002e-10
9 8 -8.8541900000000002e-10
9 9 1.7708380000000000e-09
9 10 -8.8541900000000002e-10
10 9 -8.8541900000000002e-10
10 10 1.7708380000000000e-09
10 11 -8.8541900000000002e-10
11 11 1.7708375635183248e-09
```

```
%%MatrixMarket matrix coordinate real general
11 11 29
1 1 2*eps0/dx^2
2 1 -eps0/dx^2
2 2 2*eps0/dx^2
2 3 -eps0/dx^2
3 2 -eps0/dx^2
3 3 2*eps0/dx^2
3 4 -eps0/dx^2
4 3 -eps0/dx^2
4 4 2*eps0/dx^2
4 5 -eps0/dx^2
5 4 -eps0/dx^2
5 5 2*eps0/dx^2
5 6 -eps0/dx^2
6 5 -eps0/dx^2
6 6 2*eps0/dx^2
6 7 -eps0/dx^2
7 6 -eps0/dx^2
7 7 2*eps0/dx^2
7 8 -eps0/dx^2
8 7 -eps0/dx^2
8 8 2*eps0/dx^2
8 9 -eps0/dx^2
9 8 -eps0/dx^2
9 9 2*eps0/dx^2
9 10 -eps0/dx^2
10 9 -eps0/dx^2
10 10 2*eps0/dx^2
10 11 -eps0/dx^2
11 11 2*eps0/dx^2
```

$$\left(\frac{-\epsilon_0}{\Delta x^2}\right)\begin{bmatrix} -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 \end{bmatrix}$$

Assuming Ax=b, x and b are esSolve vectors

esSolveWriteVector.mtx (b)

%%MatrixMarket matrix array real general
11 1

$$= \frac{2\epsilon_0}{\Delta x^2} \cdot \phi^{left}$$

0.0000000000000000e+00
6.1803398874989481e+00
1.1755705045849464e+01
1.6180339887498949e+01
1.9021130325903069e+01
2.0000000000000000e+01
1.9021130325903069e+01
1.6180339887498949e+01
1.1755705045849465e+01
6.1803398874989499e+00

$$= 20 \sin\left(\frac{\pi x_j}{L}\right) = \rho_j$$

1.7708375635183248e-09

$$= \frac{2\epsilon_0}{\Delta x^2} \cdot \phi^{right}$$

esSolveReadVector.mtx (x)

%%MatrixMarket matrix array real general
11 1

0.0000000000000000e+00 $= \phi^{left}$
7.1308064483412903e+10
1.3563599878269949e+11
1.8668693648958243e+11
2.1946365612852356e+11
2.3075774401243900e+11
2.1946365612872348e+11
1.8668693648998233e+11
1.3563599878329944e+11
7.1308064484212875e+10

$= \phi_j$

1.0000000000000000e+00 $= \phi^{right}$

Regroup and Review

So far, we have:

-solved the discrete 1D Poisson equation ‘by hand’ and looked at the matrix and the vectors involved in that process

-looked at how VSim builds this matrix and these vectors with a FieldUpdater (of kind linearSolveUpdater), using MatrixFiller and StencilElement and LinearSolver blocks

-seen how to modify the .in file

-seen how to examine the matrix and vectors VSim builds.

But:

-most interesting problems are not 1D

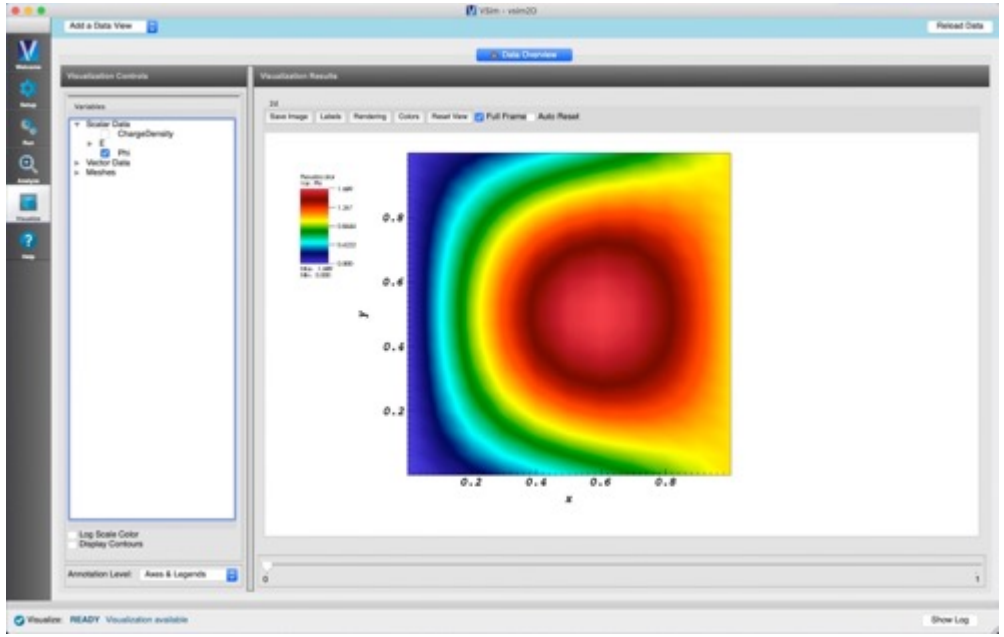
-most interesting problems involve particles, complicated geometries, and/or complicated boundary conditions

Let’s add some interesting features to our input file, and see how the .in file changes.

Moving to 2D

Let's copy the simulation we had before into a new simulation, and add:

Parameters
LY = 1
NY = 15
RHOZERO = 2.0e-10



SpaceTimeFunctions
RHOxt=RHOZERO*sin(PI*x/LX)*sin(PI*y/LY)
LINEARPHIxt=VLEFT+(VRIGHT-VLEFT)*x/LX

FieldBoundaryConditions
TOPBC: Dirichlet, LINEARPHIxt, upper y
BOTTOMBC: Dirichlet, LINEARPHIxt, lower y

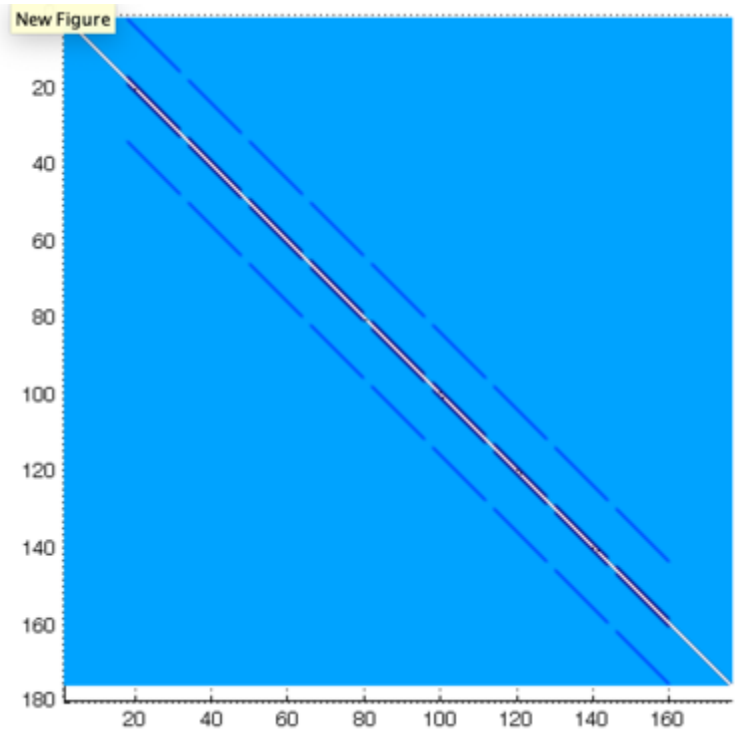
Basic Settings
dimensionality = 2

Grid
yMin=0
yMax=LY
yCells=NY

save as vsim2D.sdf

Matrix is larger, no longer tridiagonal

Now 176 x 176 [176 = 11*16 = (NX+1)*(NY+1)] and band-structured



ρ and ϕ arrays are now representing 2D quantities in a vector, e.g.

$$\begin{bmatrix} \rho_{1,1} \\ \vdots \\ \rho_{1,N} \\ \rho_{2,1} \\ \vdots \\ \rho_{2,N} \\ \vdots \\ \rho_{M,N} \end{bmatrix}$$

The same approach generalizes to 3D also; we will have large sparse matrices.

In general this 2D input file looks pretty similar to the 1D version.

Additional StencilElements relevant in 2D/3D

Δy^2

Only if ≥ 2D

+1 cell in y

Typical stencil elements:

```
<STFuncStencilElement phi_npy>
 value = 225.0
 minDim = 2
 cellOffset = [0 1 0]
 functionOffset = [0. 0.5 0.]
 rowFieldIndex = 0
 columnFieldIndex = 0
</STFuncStencilElement>

...

<STFuncStencilElement phi_nmz>
 value = 144.0
 minDim = 3
 cellOffset = [0 0 -1]
 functionOffset = [0. 0. -0.5]
 rowFieldIndex = 0
 columnFieldIndex = 0
</STFuncStencilElement>
```

Δz^2

Only if ≥ 3D

-1 cell in z

In 2D, general matrix row is

$$coefficient \cdot [\cdots \quad 0 \quad phi_{nmy} \quad \cdots \quad phi_{nmx} \quad (phi_{dxm}+phi_{dxp}+phi_{dym}+phi_{dyp}) \quad phi_{npx} \quad \cdots \quad phi_{npy} \quad 0 \quad \cdots]$$

Adding GridBoundary geometric features

Let's modify our simulation some more, to add geometric features:

Materials

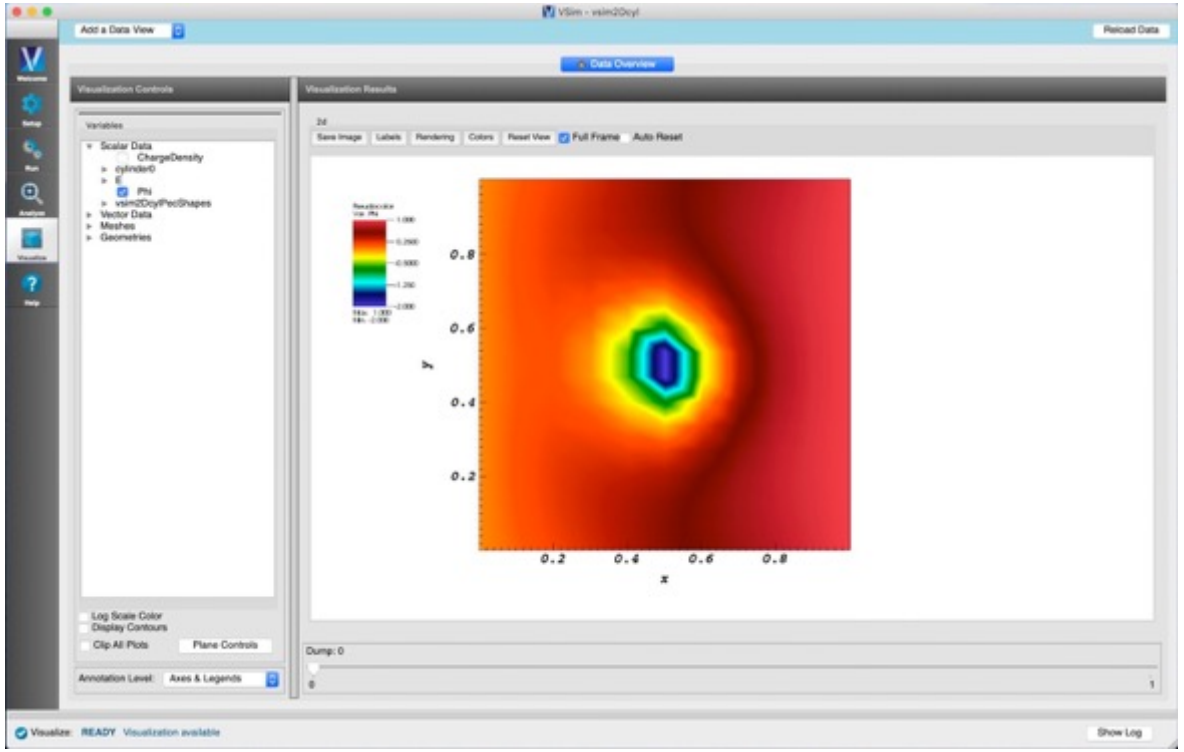
PEC: add to simulation

Geometries

CSG: Add Primitive: cylinder
material = PEC
length = 0.5
radius = 0.1
x position = 0.5
y position = 0.5
z position = -0.25
axis direction x = 0.0
axis direction y = 0.0
axis direction z = 1.0

FieldBoundaryConditions

CYLBC: Dirichlet, on cylinder, -2.0 V



save as vsim2Dcyl.sdf

New: Material and GridBoundary blocks

```
<EmMaterial PEC>
 kind = conductor
 resistance = 0.0
</EmMaterial>

<GridBoundary cylinder0>
 kind = gridRgnBndry
 calculateVolume = 1
 dmFrac = 0.5
 polyfilename = cylinder0.stl
 flipInterior = True
 scale = [1.0 1.0 1.0]
 printGridData = False
 mappedPolysfile = cylinder0_mapped.stl
</GridBoundary>
```

See documentation…
https://www.txcorp.com/images/docs/vsim/latest/VSimReferenceManual/vsimComposerMaterials.html
and
https://www.txcorp.com/images/docs/vsim/latest/VSimReferenceManual/blocks_gridboundary.html

New: GridBoundary MatrixFillers

```
<MatrixFiller CYLINDERFiller>
 kind = nodeStencilFiller
 gridBoundary = cylinder0
 rowInteriorosity = [cutByBoundary outsideBoundary]
 colInteriorosity = [cutByBoundary outsideBoundary]
 component = 0
 minDim = 1
 lowerBounds = [1 1 1]
 upperBounds = [10 15 12]

 <StencilElement ident>
  value = 5.7552220814345554e-09
  minDim = 1
  cellOffset = [0 0 0]
  rowFieldIndex = 0
  columnFieldIndex = 0
 </StencilElement>

</MatrixFiller>
```

```
<VectorWriter CYLINDERWriter>
 kind = stFuncNodeVectorWriter
 gridBoundary = cylinder0
 minDim = 1
 lowerBounds = [1 1 1]
 upperBounds = [10 15 12]
 component = 0
 interiorosity = [cutByBoundary outsideBoundary]

 <STFunc function>
  kind = expression
  expression = -2.0
 </STFunc>

 scaling = 5.7552220814345554e-09
</VectorWriter>
```

See documentation…

As before, we could go and look at the matrix again, to see how these operations changed it, and get a sense for what VSim is doing behind-the-scenes.

Adding particles to an input file

- Instead of doing this through the visual setup, let's just open an example and test our developing .in-file-reading skills.
- File > New From Example > VSim for Plasma Discharges > Capacitively Coupled Plasma > Turner Case 2
- I'll show a quick movie of this discharge so that you have a sense for what we'll be looking at: available here: http://nucleus.txcorp.com/~tgjenkins/movies/ShortCCPmovie.mov
- Neutral gas is contained between two parallel plates; one plate is grounded and the other biased with RF. The motion of free electrons creates plasma between the plates, and the formation of plasma sheaths is observed. The long-time steady state of the discharge is a balance between collisional ionization (source) and wall losses (sink).

Looking at the Turner .in file - ScalarDepositors

- Some familiar things: Fields, FieldUpdaters, UpdateSteps, MultiFields, etc.
- Some new things: ScalarDepositor, Species, Fluid, History, collisional physics, etc.

ScalarDepositors act like “buckets” that collect particle charge on the simulation grid.

```
<ScalarDepositor ChargeDensityDep>
 kind = areaWeighting
 depField = esMultiField.ChargeDensity
</ScalarDepositor>
```

charge-collecting algorithm

where to store collected charge

When all particles have been put into the bucket, its contents are then put into the specified depField.

Looking at the Turner .in file - Species

- Some familiar things: Fields, FieldUpdaters, UpdateSteps, MultiFields, etc.
- Some new things: ScalarDepositor, Species, Fluid, History, collisional physics, etc.

Species = almost everything having to do with the particle-in-cell aspects of VSim. Details beyond the scope of this already-long talk, but the principles are the same – *nested block structures* that describe objects and their interactions with other objects.

```
<Species electrons>
 kind = nonRelBoris
 charge = -1.6021766208e-19
 mass = 9.10938215e-31
 ...
 <ParticleSource particleLoaderE>
  ...
  <PositionGenerator posGen>
   ...
  </PositionGenerator>
  <VelocityGenerator velGen>
   ...
  </VelocityGenerator>
 </ParticleSource>
 <ParticleSink leftElecAbsorber>
  ...
 </ParticleSink>
</Species>
```

Looking at the Turner .in file - History

- Some familiar things: Fields, FieldUpdaters, UpdateSteps, MultiFields, etc.
- Some new things: ScalarDepositor, Species, Fluid, History, collisional physics, etc.

History blocks create a record of various physics events on a *per-timestep* basis, rather than on a *per-dump-step* basis (e.g. current entering a wall, number of particles in the simulation, etc.)

```
<History numElec>
 kind = speciesNumberOf
 species = [electrons]
</History>

<History leftIonCurr>
 kind = speciesCurrAbs
 species = [He1]
 ptclAbsorbers = [leftIonAbsorber]
</History>
```

count the number of macroparticles in a species

specify the species

measure absorbed particle current from a species

specify the species

specify the absorbing region

Exercises to test your text-file reading skills

- Look at the Turner.in file and see if you can identify how a species identifies the electric and magnetic fields that its particles respond to.
- Look in the VSim documentation at the different “kinds” of particle species (besides nonRelBoris) that are available. Could a nonRelES species work equally well for the Turner example?
- See if you can determine how particles are loaded in velocity space in the Turner.in input file. How would you add a mean flow to the particles?
- See if you can add a history block to Turner.in that records various properties (kinetic energy, velocity, loss time, etc.) of electrons that strike the left wall of the simulation and are lost.

Useful resources: reminder

- VSim online documentation:
https://www.txcorp.com/images/docs/vsim/latest/VSimDocumentation.html

- Slides for this talk:
http://nucleus.txcorp.com/~tgjenkins/pres/TWSSTalk2020.pdf

- Download page for the VSim input files I used in this talk:
http://nucleus.txcorp.com/~tgjenkins/TWSS2020.html

- Also potentially of interest: Slides for other VSim talks I've given in the past (visualization, plasma sheath modeling, RF antenna simulations, CCPs, etc.):
http://nucleus.txcorp.com/~tgjenkins/informal.html

Summary/Overview

- This talk was only a top-level view of the kinds of things you'll see if you edit text-based VSim input files… it's certainly possible to dig deeper.
- But if you're comfortable with the idea of block structures, and with digging into the documentation, I've hopefully given you enough information that you can start to tackle your own problems.
- Nevertheless, it's good to ask questions if you get stuck - please feel free to do so as you're figuring this stuff out. There are quite a few "power-users" of VSim who have probably had to wrestle with many of the problems you'll run into.
- Thanks for your attention!