Semiconductor Device Simulation Using DEVSIM

Juan Sanchez

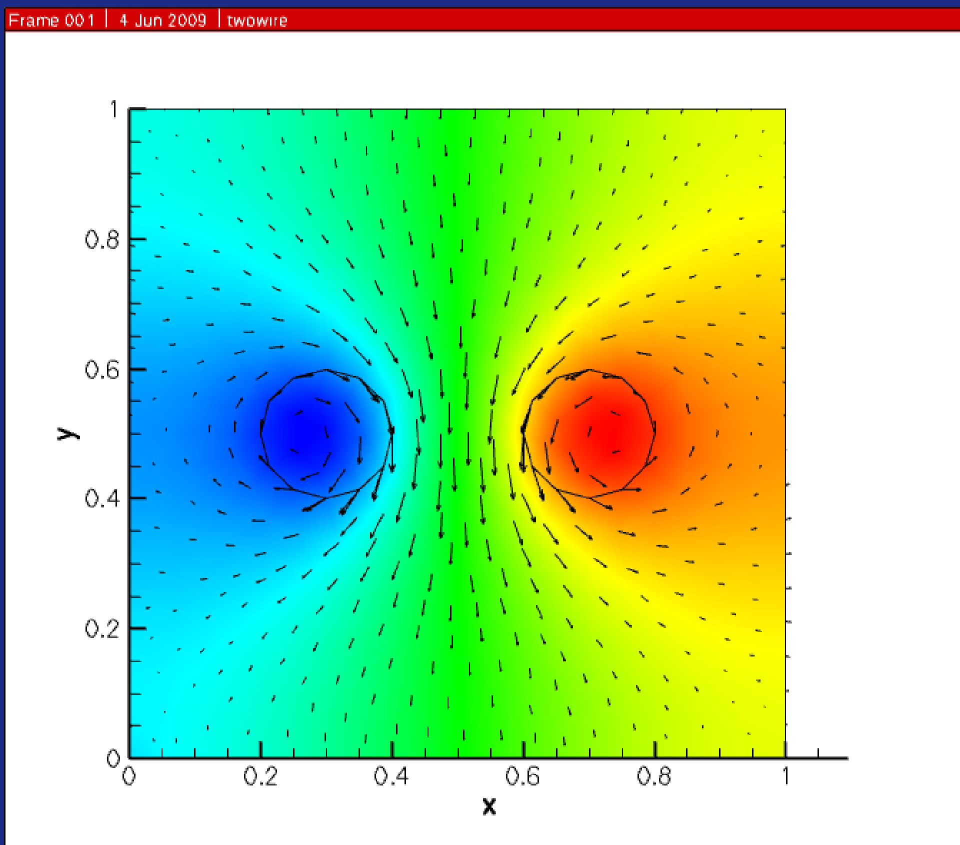
July 2, 2018

Introduction

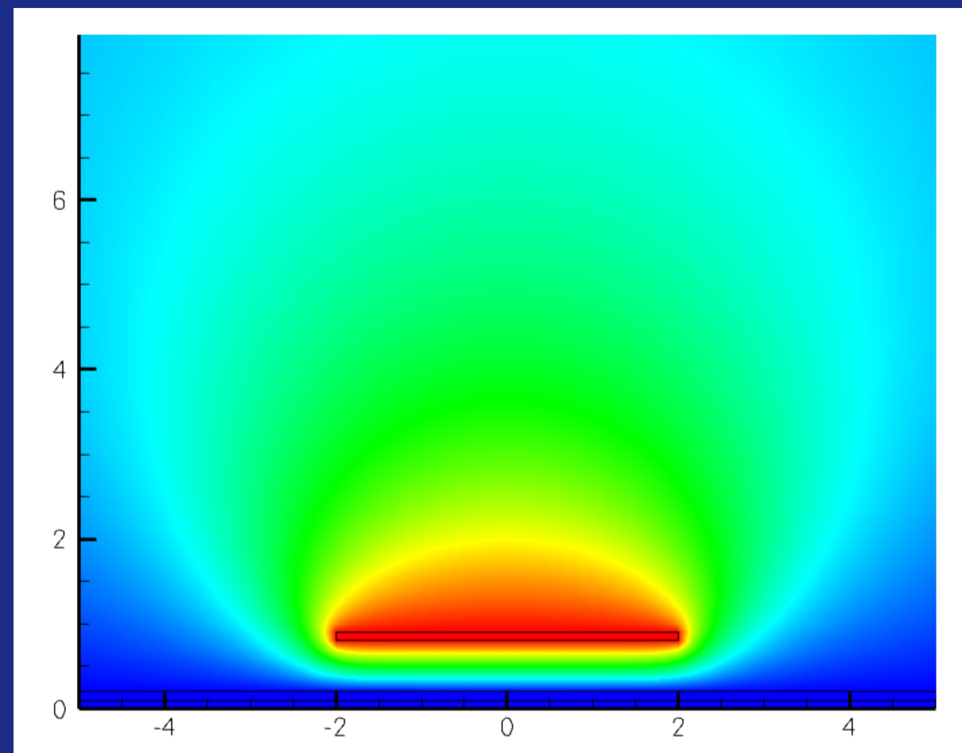
- PDE semiconductor device simulator
- Finite volume method
- Solves 1D, 2D, and 3D structures
- External meshing tools or internal mesher
- Symbolic model evaluation
- Visualization using standard output formats

Examples

Magnetic Potential



Capacitance



Device Equations

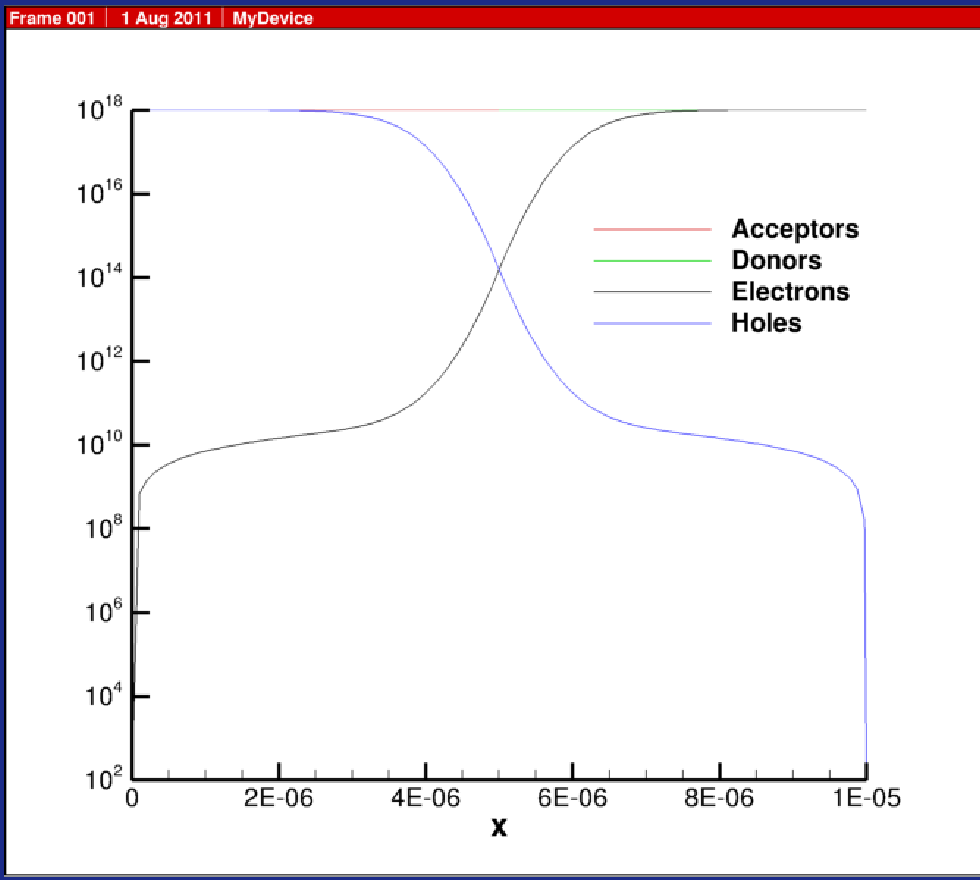
- Drift-diffusion equations

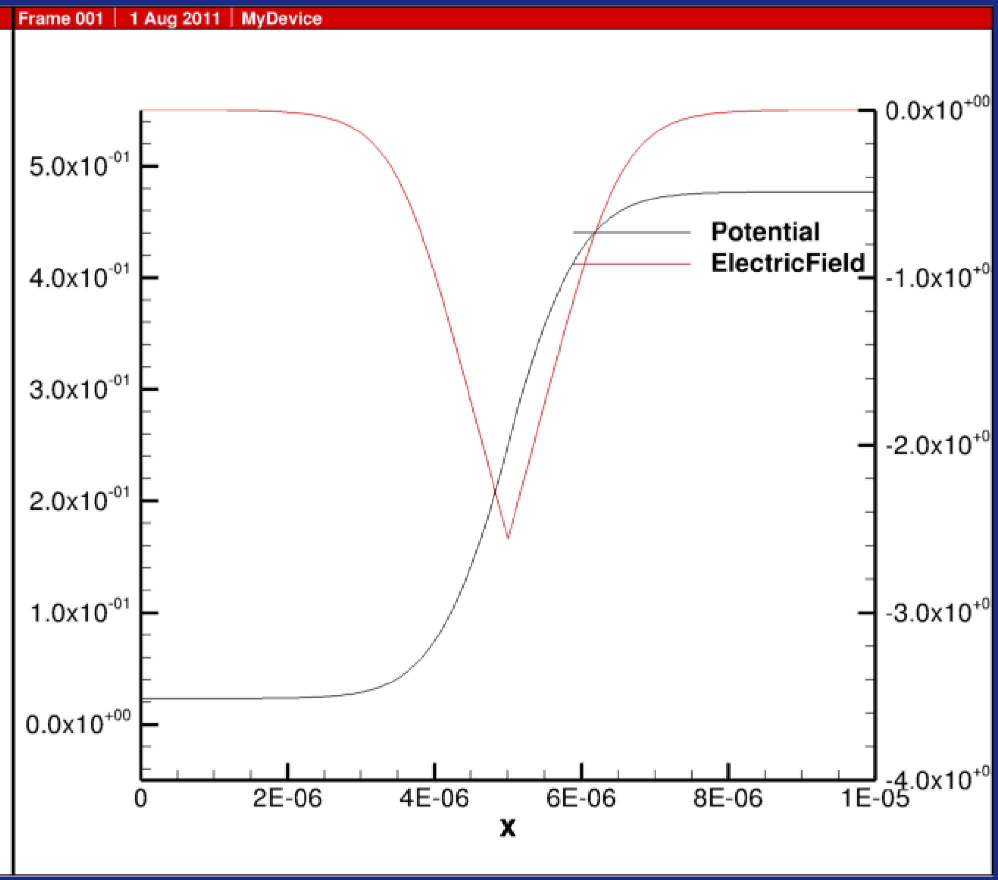
$$\nabla^2 \varphi = q\left(p - n + N_D - N_A\right) \quad \text{(Poisson)}$$

$$\frac{\partial n}{\partial t} = \frac{1}{q}\nabla \cdot \vec{J}_n + G_n - R_n \quad \text{(Electron Continuity)}$$

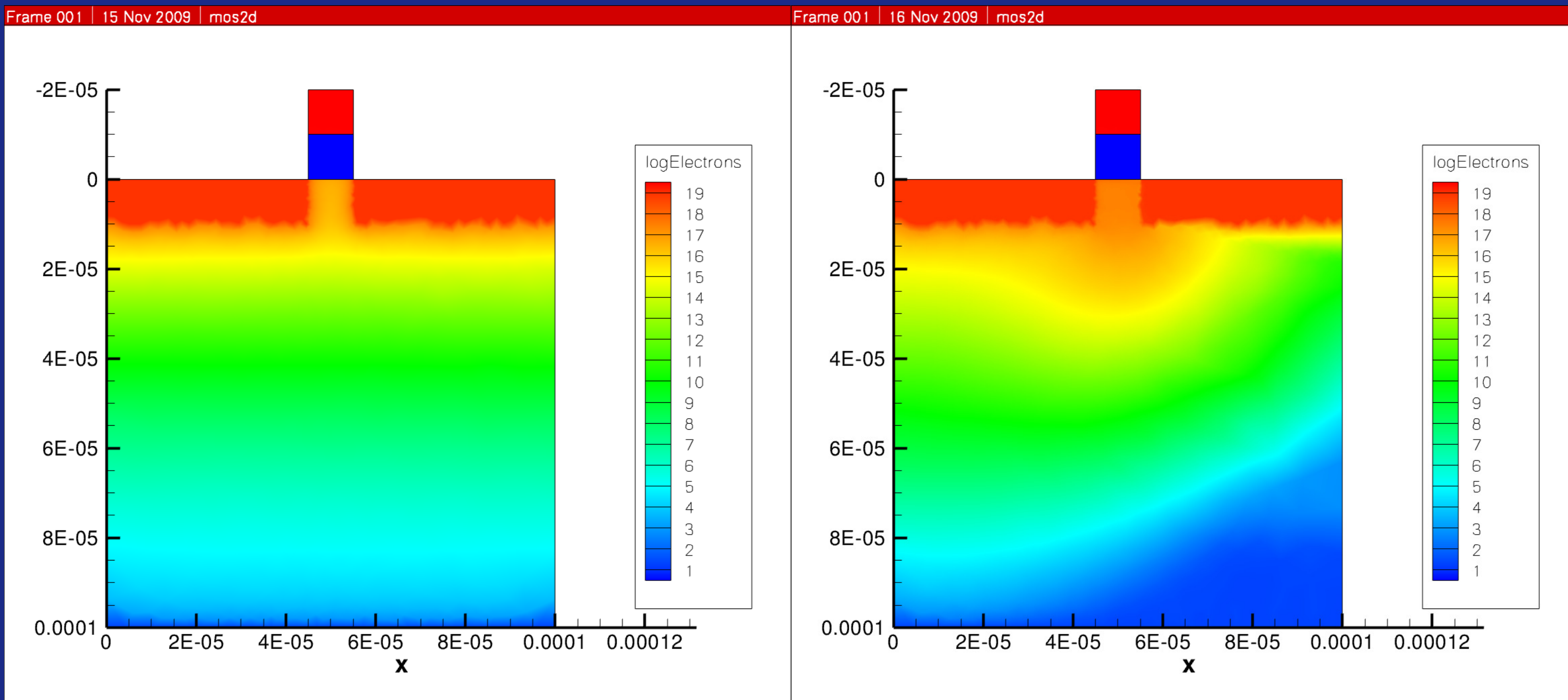
$$\frac{\partial p}{\partial t} = -\frac{1}{q}\nabla \cdot \vec{J}_p + G_p - R_p \quad \text{(Hole Continuity)}$$

Example – 1D Diode

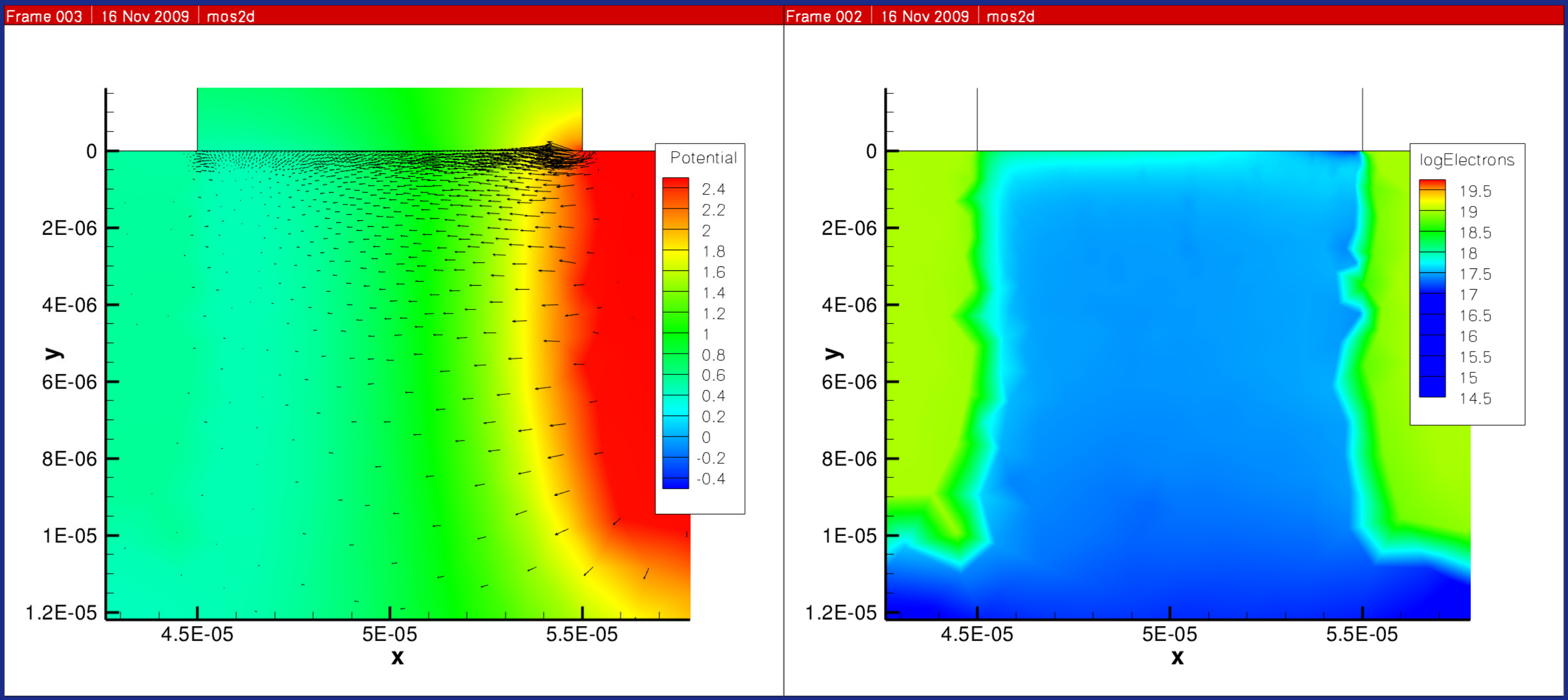




Example – 2D MOSFET



Example – 2D MOSFET



Introduction

- Project started in 2008
- Open source since 2013 `https://devsim.org`
- C++ using STL, C++-11, and templates
- Platform Agnostic (Linux, OS X, Windows)
- Uses Python scripting to set up equations and control simulation
- Approximately 64,000 lines of code

  `https://www.openhub.net/p/devsim`

Architecture – Analysis

- Nonlinear simulation
  - dc
  - transient
- Linear analysis
  - small-signal ac
  - sensitivity (impedance field)
  - noise

Architecture – Scripting

- Models implemented using scripting
  - Faster development cycle
  - Design for efficiency
- Symbolic differentiation
  - Faster development time
  - Add derivatives w.r.t. new variables
  - Common subexpression elimination

Architecture – Python

- well defined and consistent
- avoids domain specific languages with limited debugging
- provides users more control
- has numerous libraries for analysis and visualization

Architecture – Numerics

- BLAS and LAPACK
  - Used for dense matrix and vector operations, geometric processing
  - Optimized for most platforms
  - Called by sparse matrix factorization
- SuperLU, MKL Pardiso used for sparse matrix factorization
- Iterative Math Library used for GMRES

SYMDIFF

- Symbolic differentiation library
- Open source `https://symdiff.org`
- String based approach with dynamic binding of names to referred quantities
  - Constants
  - Independent variables
  - Models

SYMDIFF – Parser

- Uses rules of precedence and associativity
- Has `simplify` algorithm to reduce cost

```
<<<< diff(a + b + c^2, c)
(2 * c)
<<<< diff(x^x, x)
(((x * (x^(-1))) + log(x)) * (x^x))
<<<< simplify(diff(x^x,x))
((1 + log(x)) * (x^x))
```

SYMDIFF – User functions

- Defining functions requires specification of new function and derivatives w.r.t. each named variable argument

```
> define(sqrt(x),0.5 * x^(-0.5))
sqrt(x)
> diff(sqrt(x*y),y)
((0.5 * ((x * y)^(-0.5))) * x)
```

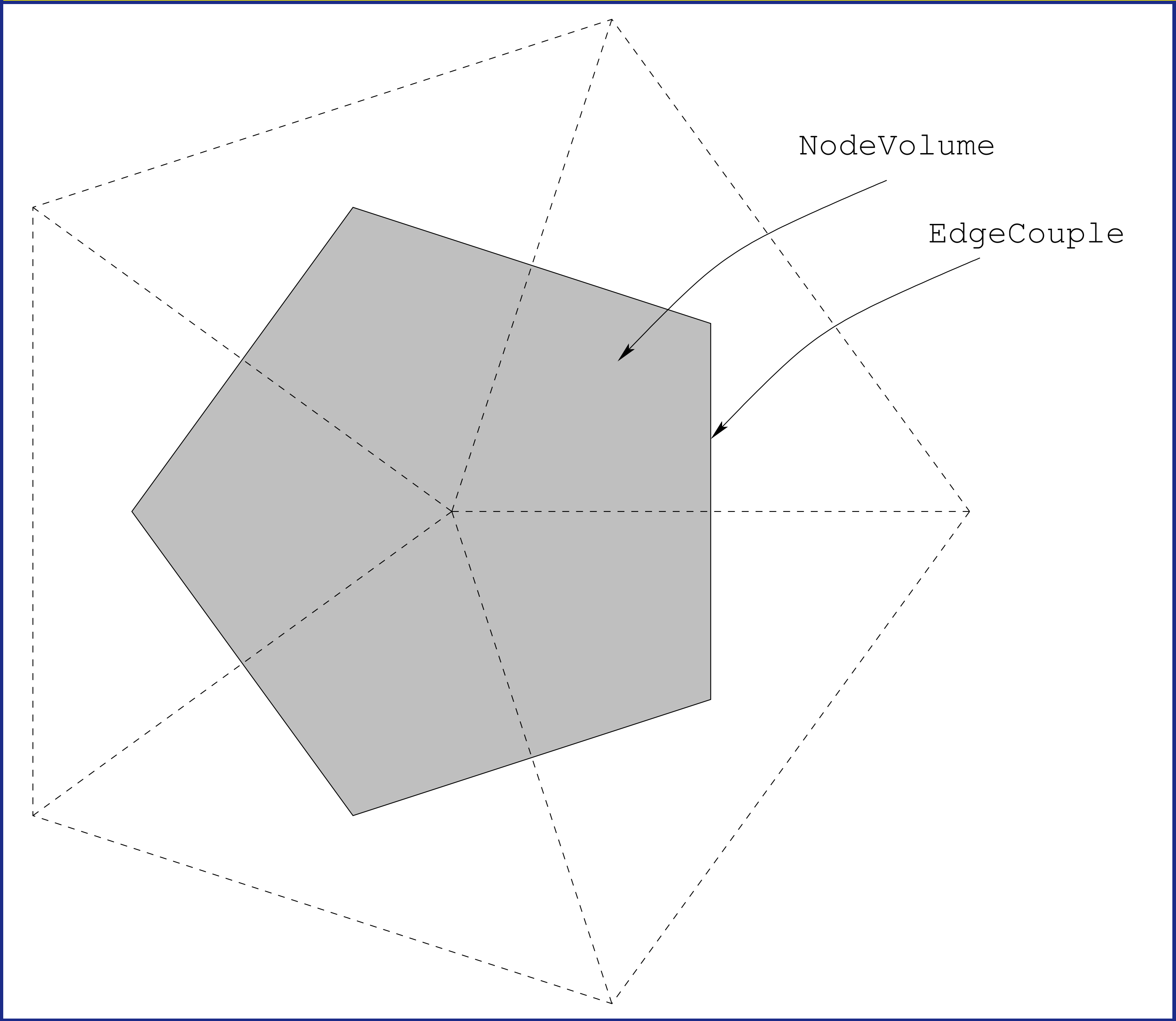
SYMDIFF – Models

- Models allow
  - creation of new PDEs
  - hierarchy for sub-expression elimination
  - ability to specify or generate derivatives
- Models dynamically bound by name
  - `diff(Model,x)` is `Model:x`

Element Assembly

- Expressions evaluated at run time
- Symbolic derivatives of models for Jacobian assembly
- Assembles bulk, interface, and contact equations
- Circuit boundary conditions

Node Models

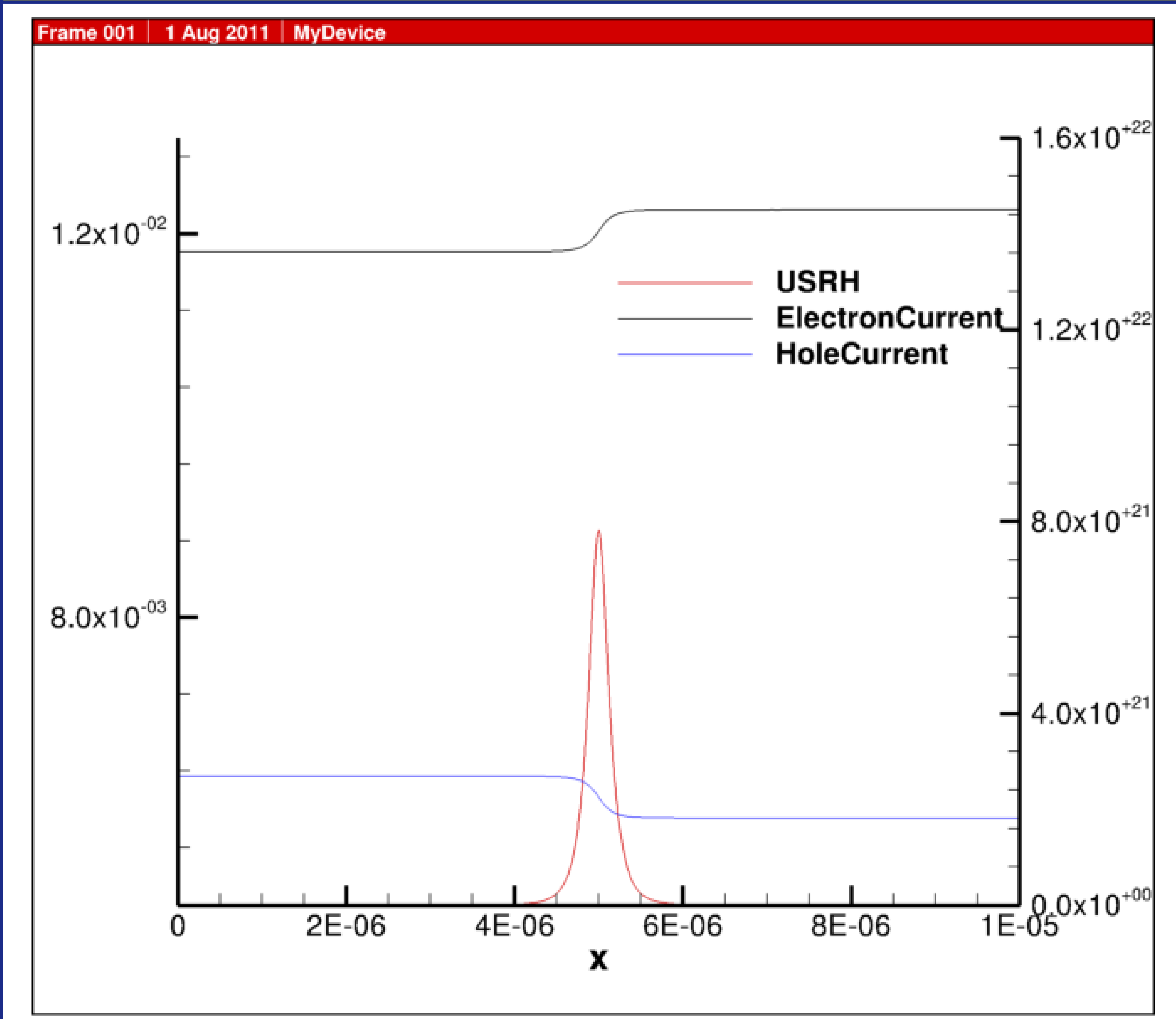


Node Models – Shockley Read Hall

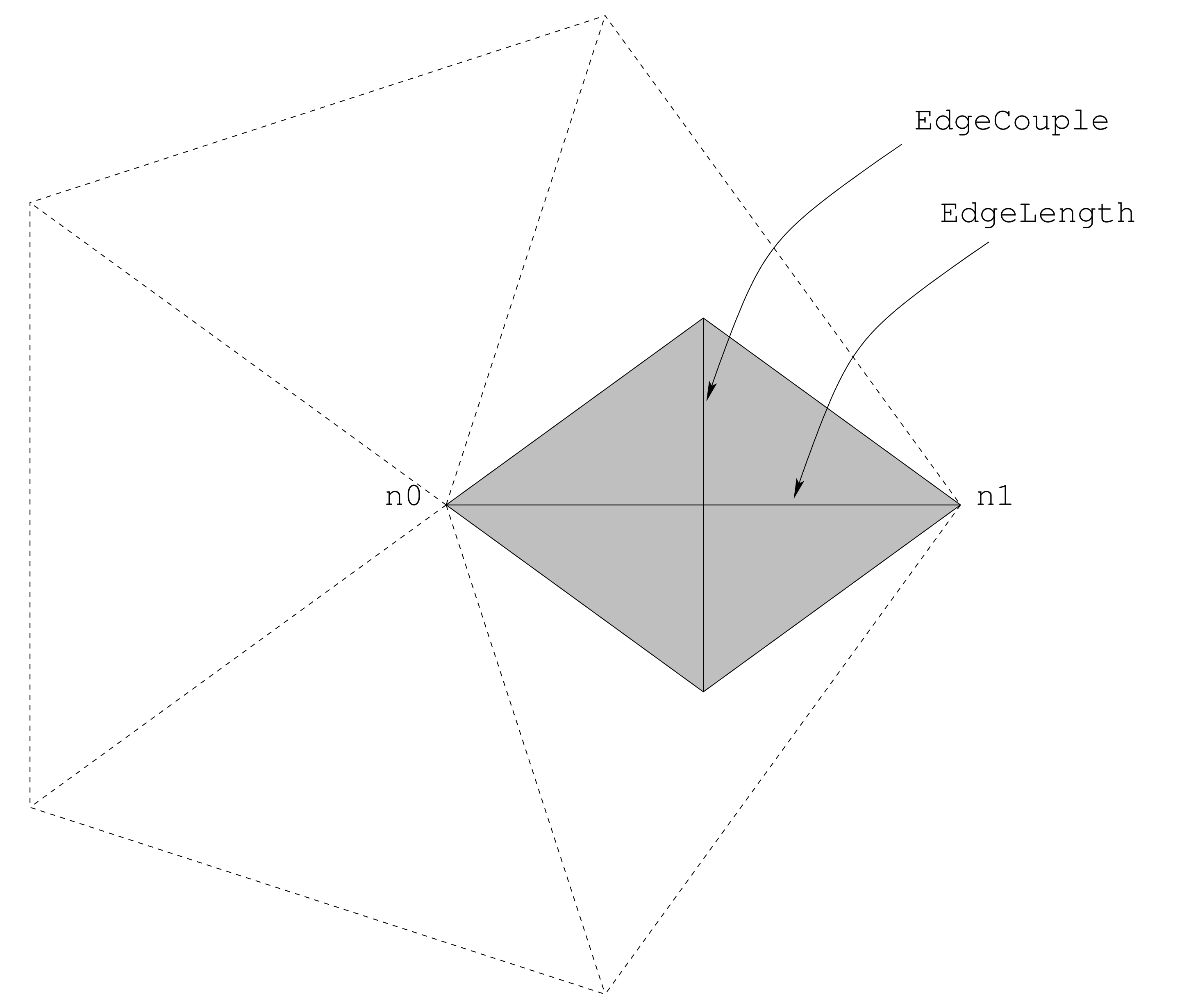
$$U_{\mathsf{SRH}} = \frac{np - n_i^2}{\tau_p(n + n_1) + \tau_n(p + p_1)}$$

```
USRH="(Electrons*Holes - n_i^2)/ \
  (taup*(Electrons + n1) + taun*(Holes + p1))"
Gn = "-ElectronCharge * USRH"
Gp = "+ElectronCharge * USRH"
NodeModel("USRH", USRH)
NodeModel("ElectronGeneration", Gn)
NodeModel("HoleGeneration", Gp)
for i in ("Electrons", "Holes"):
  NodeModelDerivative("USRH", USRH, i)
  NodeModelDerivative("Gn", Gn, i)
  NodeModelDerivative("Gp", Gp, i)
```

Node Models – Shockley Read Hall



Edge Models

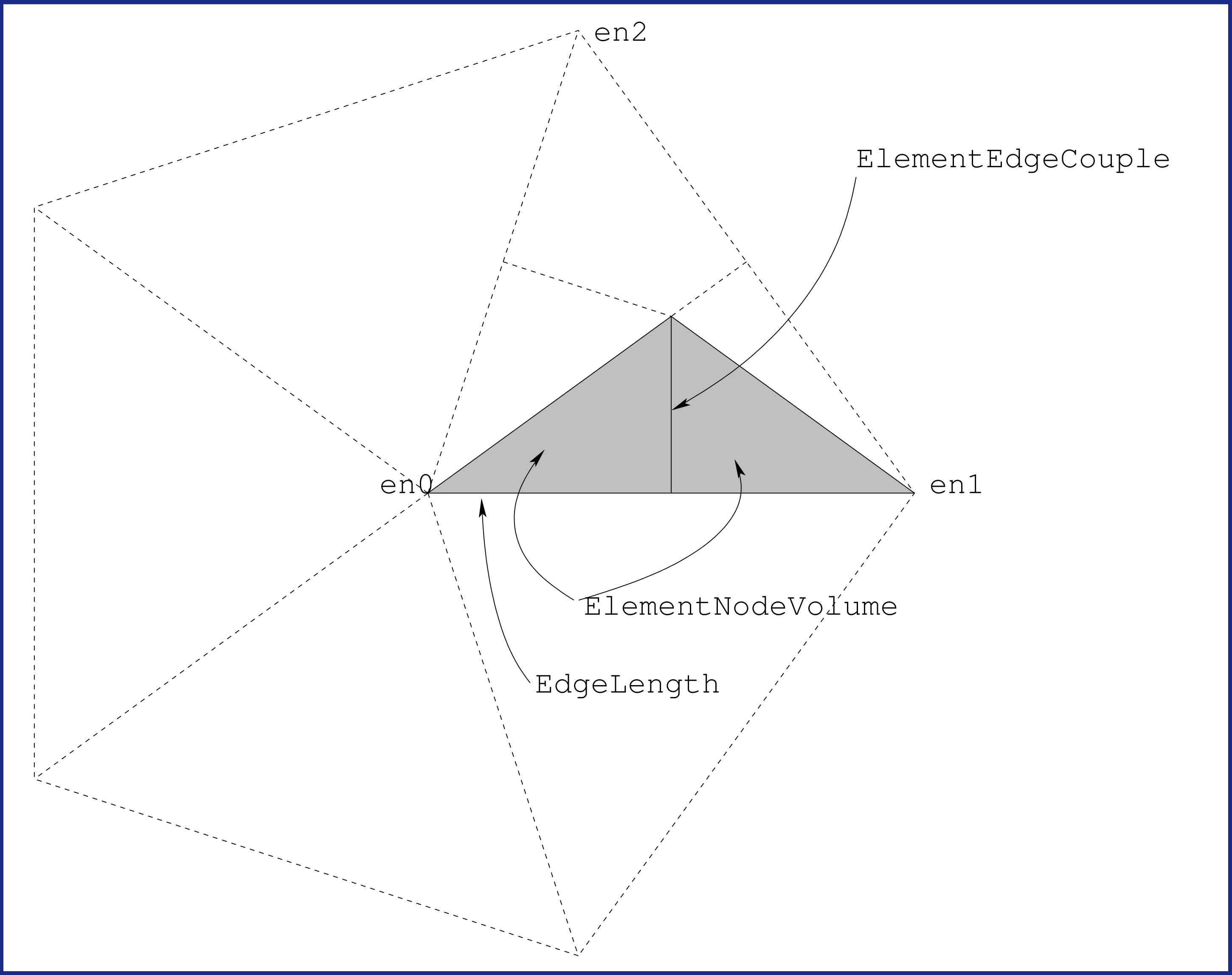


Edge Models

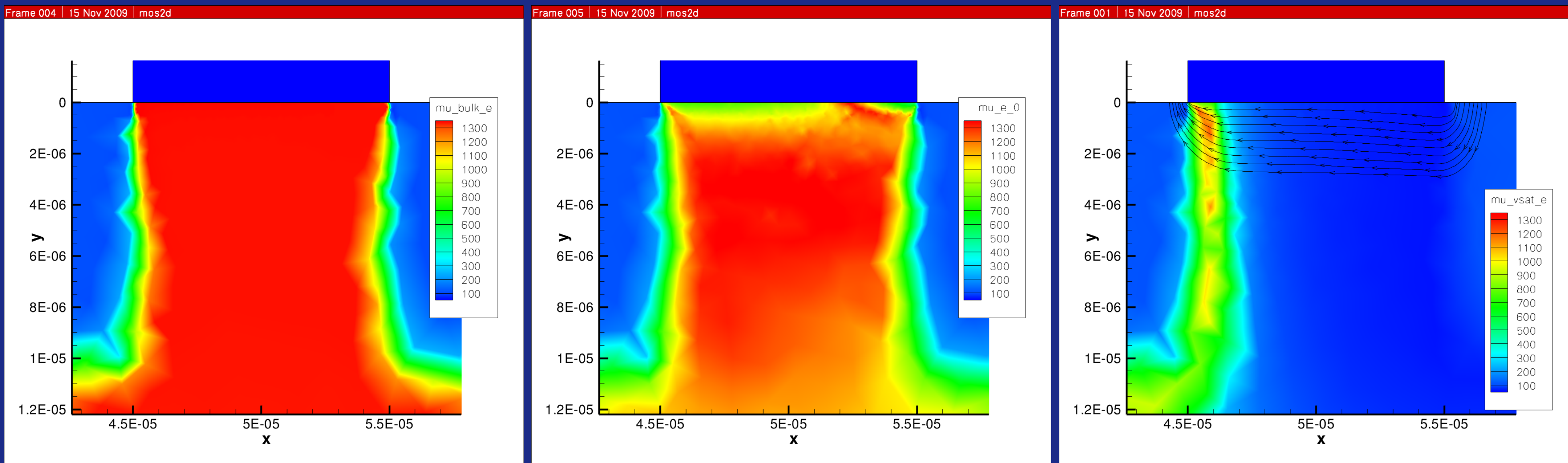
- Electric field ($\mathscr{E}$) w.r.t potential (φ)

```
edge_model(device=device, region=region,
  name='ℰ',
  equation='(φ@n0 - φ@n1)*EdgeInverseLength')

edge_model(device=device, region=region,
  name='ℰ:φ@n0',
  equation='EdgeInverseLength')

edge_model (device=device, region=region,
  name='ℰ:φ@n1',
  equation='-EdgeInverseLength')
```

Element Edge Models



2D MOSFET Mobility

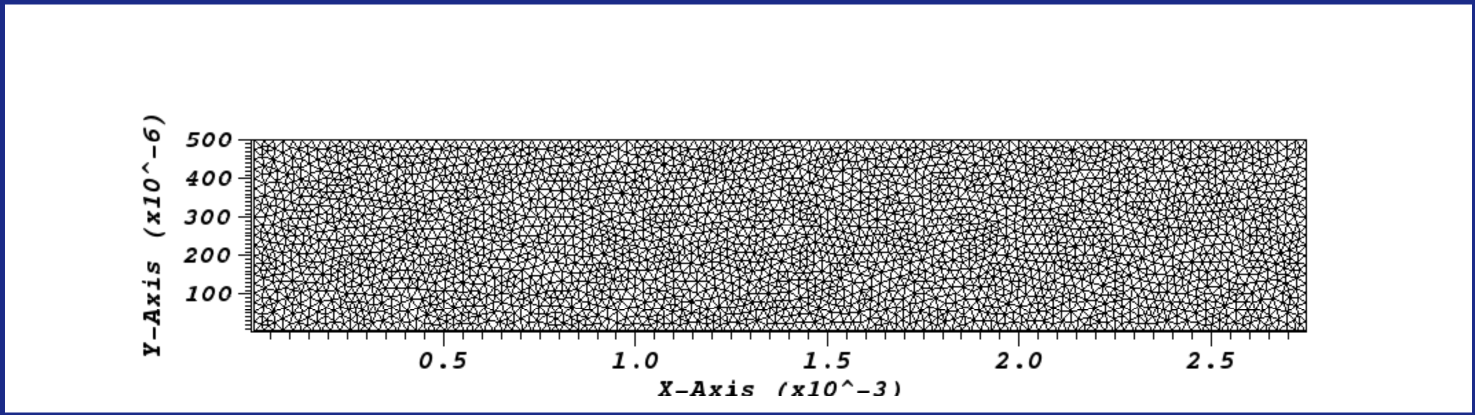


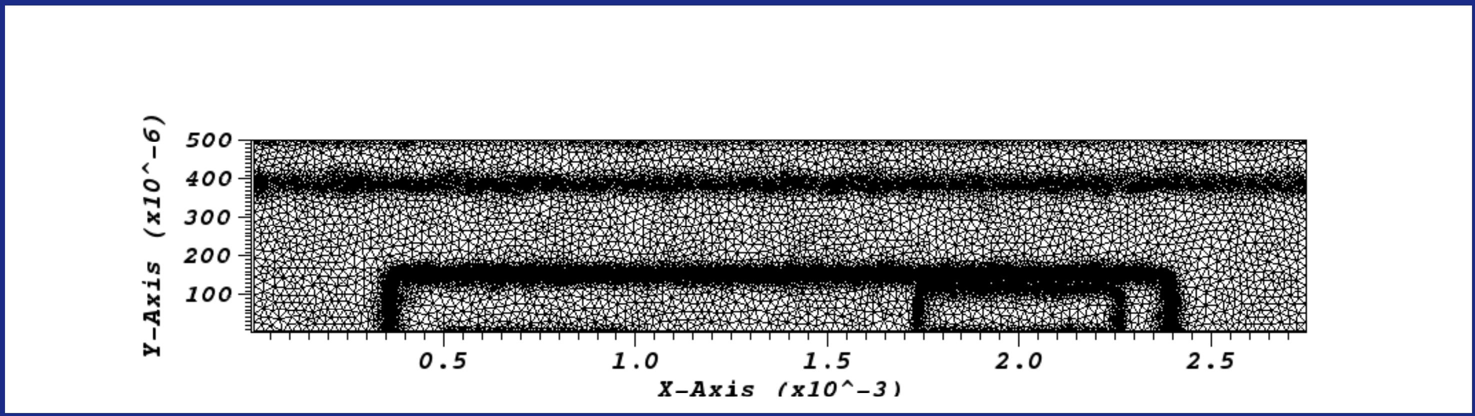
- Element models are used to simulate mobility with respect to electric field normal and perpendicular to current flow

BJT Example

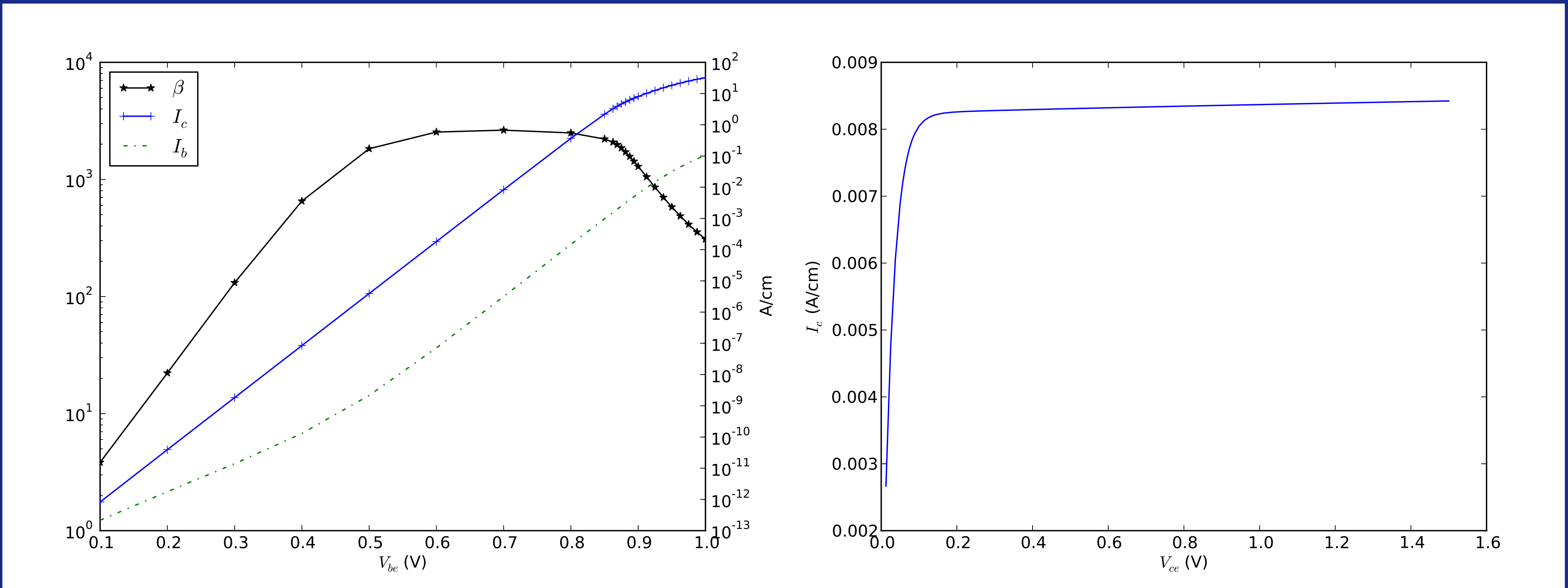
Available

`https://github.com/devsim/devsim_bjt_example`

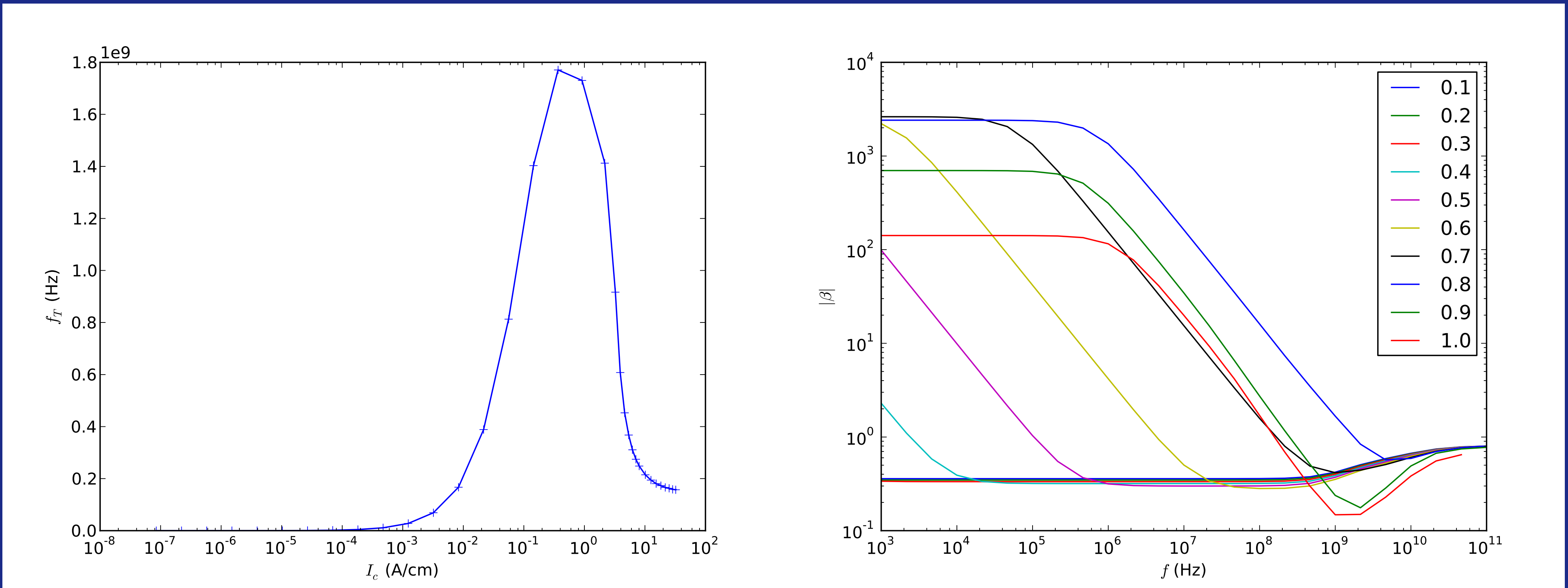




BJT – DC Analysis



BJT – AC Analysis



Density Gradient

- Quantum correction method for carrier density near interfaces
- Carrier quantization effects

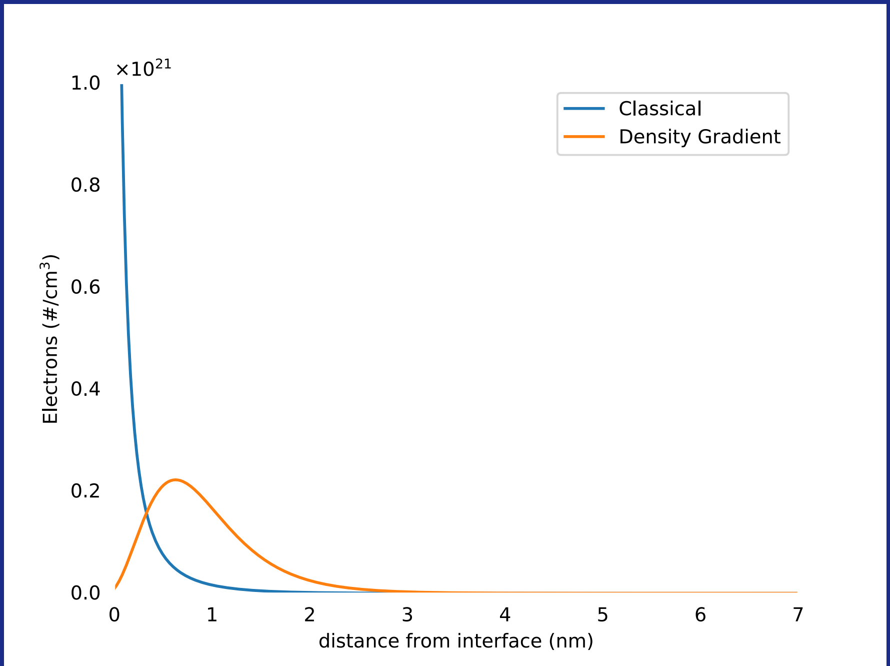
$$\Lambda_e = -b_n \frac{\nabla^2 \sqrt{n}}{\sqrt{n}}$$

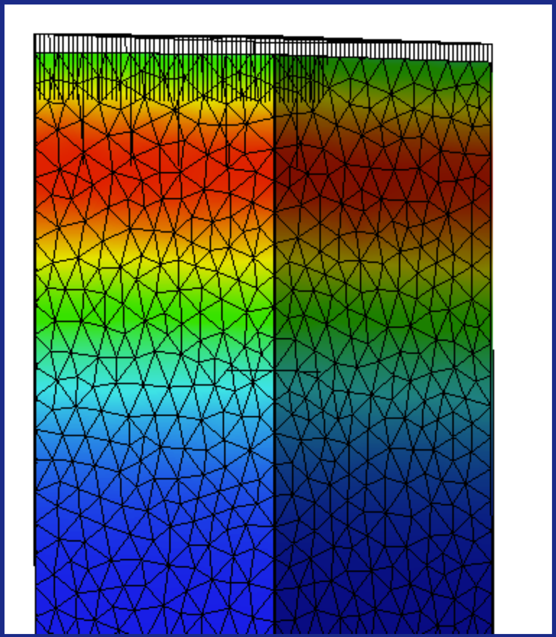
$$\frac{\nabla^2 \sqrt{n}}{\sqrt{n}} = \frac{1}{2} \left\{ \nabla^2 \log n + \frac{1}{2} \left(\nabla \log n \right)^2 \right\}$$

Using $n = \exp(u)$

$$\int \Lambda_e \partial v = -\frac{b_n}{2} \left\{ \int \nabla u \cdot \partial s + \frac{1}{2} \int \left(\nabla u \right)^2 \partial v \right\} + \frac{b_{n_{ox}}}{x_n} \sigma_{\text{int}}$$

Density Gradient





Density Gradient