J Comput Electron (2013) 12:701–721
DOI 10.1007/s10825-013-0535-y

VSP—A Quantum-Electronic Simulation Framework

Oskar Baumgartner · Zlatan Stanojevic · Klaus Schnass · Markus Karner · Hans Kosina

Published online: 28 November 2013

Abstract The Vienna Schrödinger-Poisson (VSP) simulation framework for quantum-electronic engineering applications is presented. It is an extensive software tool that includes models for band structure calculation, self-consistent carrier concentrations including strain, mobility, and transport in transistors and heterostructure devices. The basic physical models are described. Through flexible combination of basic models sophisticated simulation setups for particular problems are feasible. The numerical tools, methods and libraries are presented. A layered software design allows VSP's existing components such as models and solvers to be combined in a multitude of ways, and new components to be added easily. The design principles of the software are explained. Software abstraction is divided into the data, modeling and algebraic level resulting in a flexible physical modeling tool. The simulator's capabilities are demonstrated with real-world simulation examples of tri-gate and nanoscale planar transistors, quantum dots, resonant tunneling diodes, and quantum cascade detectors.

Keywords VSP · Schrödinger · Quantum-Mechanical Simulation · Nano-Electronic Devices · Numerical Methods

The first two authors contributed equally to this work.

Oskar Baumgartner (✉) · Zlatan Stanojevic · Hans Kosina
Institute for Microelectronics, TU Wien
Gußhausstraße 27-29/E360, 1040 Wien, Austria
Tel.: +43-1-58801-36015
e-mail: {baumgartner|stanojevic|kosina}@iue.tuwien.ac.at

Klaus Schnass · Markus Karner
Global TCAD Solutions GmbH
Landhausgasse 4/1a, 1010 Wien, Austria
Tel.: +43-664-2162901
e-mail: {k.schnass|m.karner}@globaltcad.com

1 Introduction

Several software packages exist dealing with some aspects of quantum-electronic computation. Early codes began to appear in the late 1990s as one-dimensional Schrödinger-Poisson codes for the analysis of heterostructures, gate stacks, Schottky contacts, or surfaces. Some of these early codes still see active use today, such as Schred [1] or "1D Poisson" by Gregory Snider [2,3]. The functionality and usability of these early tools proved to be quite limited, and during the last decade different groups have embarked on a quest for codes that allow a more general treatment of quantum-electronics and create a nanoelectronic technology computer-aided design (TCAD) software (Fig. 1). One of the well known software packages is nextnano^3 [4, 5] which solves effective mass and **k**·**p** Schrödinger equations using the finite difference method in up to three dimensions. The software also includes drift-diffusion transport and extraction of optical parameters. NEMO 5 [6] is a solver based on the tight-binding method that focuses on multi-scale physics. It provides simple models to be run at personal workstations as well as heavily parallelized, multi-million atom calculations of quantum states [7]. The tool tiberCAD [8,9] combines continuum and atomistic models for electronic transport, heat transport and optical simulations. Another finite element based solver for **k**·**p** Hamiltonians and transport in optical devices is tdkp/AQUA [10,11]. ATLAS Quantum [12] is a commercial effective mass Schrödinger solver. Sentaurus Device provides similar capabilities [13].

Based on the experiences with classic TCAD and above simulators we developed a unique basic set of design concepts. Following these rules we aim to provide the first consistent quantum-mechanical modeling platform suitable for nanoelectronic TCAD engineering

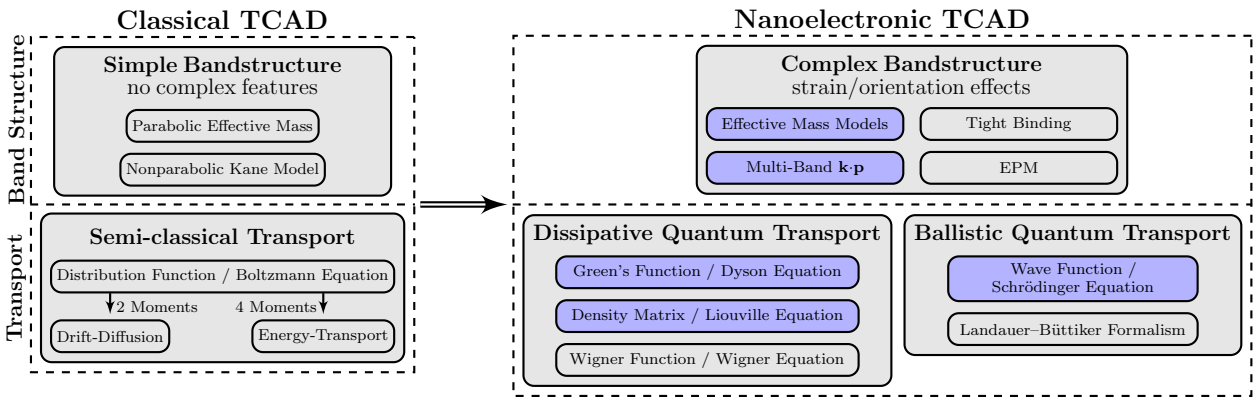


Fig. 1 Evolution from the classical TCAD approach with simple band structure models and semi-classical transport to a TCAD environment for nanoscaled devices. The implementation of an advanced band structure model including strain and orientation effects as well as quantum transport models are the major challenges in realizing a nanoelectronic TCAD simulator. Models that are in the scope of this review are marked.

applications. Here, we introduce our quantum electronic simulation framework VSP and explain our approach to nanoelectronic TCAD. VSP has been developed since 2004 [14] and a first major release was made available recently [15]. The paper is structured as follows: In Section 2 we lay out the basic design concepts that guide the development of VSP. In Section 3 we demonstrate the typical VSP-work flow that allows to rapidly define a simulation setup in a flexible way. In Section 4 we describe the architecture of VSP, its abstraction levels and software components. Section 5 summarizes the most important physical models currently available, and Section 6 gives some insight into the numerical backbone of the simulator. Finally, we introduce new ways to customize and extend VSP in Section 7 and demonstrate VSP's capabilities using representative real-world simulation examples in Section 8.

2 Design Concepts

The software design of VSP is based on five concepts which shall be elaborated in detail in the next section:

1. Flexibility
2. Automation
3. Efficiency
4. Consistency
5. Customization

From a user point of view *Flexibility* means that VSP can be conveniently and intuitively configured to perform a vast multitude of simulation tasks. Models serve as building blocks of a simulation flow. *Automation* is closely related to *Flexibility* and means providing interfaces for parameterizing the simulation flow, including scriptable input decks, parameter sweeps, and interfacing with meshing tools, device simulators etc. *Automation* also allows VSP to be used in automated device design optimization and parameter calibration. *Efficiency* is an enabling concept for *Automation*; strong emphasis has been put on *Efficiency* in the design of VSP. The model design in VSP is based on broad general approaches to solving certain classes of problems and avoids special cases. *Consistency* is ensured when dimensionality, materials, or computational methods are varied in the simulation. Finally, *Customization* allows the VSP to be extended in several ways, like adding new models, materials, or numerical libraries.

3 Work Flow and Interaction

3.1 Models and Attributes

The core design philosophy in VSP can be summarized as follows: *Everything is a model.* In the sense of VSP's design a model is an object similar to a class in computer programming. A model can be instantiated and the instance can be invoked, which may result in equations being solved, expressions evaluated and so forth. A model may have submodels for certain sub-tasks; the model may invoke its submodels during execution. Every model instance can store data relevant to the model and may expose the data, so that it can be passed to and from other model instances. Figure 2 shows the architecture of a VSP model.

Model data is organized into attributes. Three types of attributes occur: *parameters*, *properties*, and *quantities*. *Parameters* represent numerical, or non-physical entities: error tolerance, number of iteration steps, and so on. *Properties* represent concentrated physical quantities: contact voltages, subband energies, etc. *Quantities* represent distributed physical quantities – in real space (electrostatic potential, charge density) and k-space (e.g.

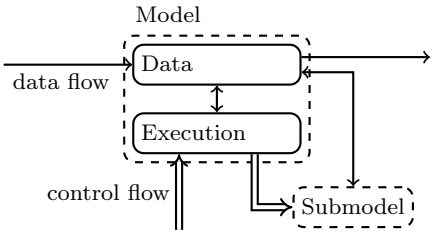
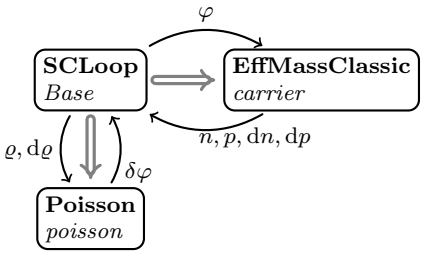


Fig. 2 Conceptual schematic of a model in VSP; solid arrows represent data flow, double arrows control flow.




```
Simulation {
  model = "SCLoop";
  Base {
    ccmodels = "EffMassClassic:carrier";
    poisson { ... }
    carrier { ... }
} }
```

Fig. 3 Upper part: model configuration with a self-consistent loop involving a Poisson model and a model for classical equilibrium carrier distributions; ⇒ indicates submodel invocation, ⟶ indicates data flow; lower part: the corresponding IPD configuration; each (sub-)model instance has its own (nested) section;

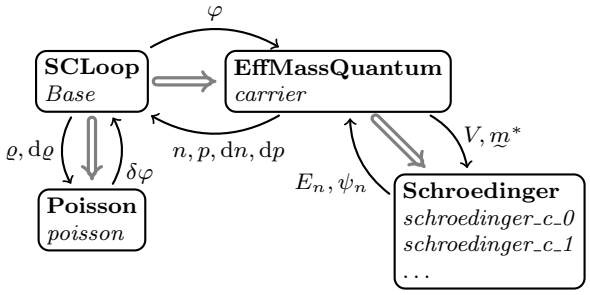
band/subband structure). All attributes have identifiers that can be used to access them. *Properties* and *quantities* have physical units.

Most models represent physical models – a Poisson solver, a carrier concentration model – but they are also used for representing numerical and logical algorithms – a self-consistent loop or nested execution.

3.2 Controlling VSP – the Input Deck Language

VSP is controlled by files written in the IPD language [16]. IPD is a hierarchically structured configuration language organized in sections. Each section may contain nested sections and variables. Variables can have physical quantities. IPD allows the use and evaluation of mathematical and logical expressions that are evaluated when needed. Any section may be derived from one or more parent sections by which it inherits all the parent sections' content.

VSP-IPDs contain three sections at the top level: `Device`, `Materials`, and `Simulation`. `Device` defines




```
Simulation {
  model = "SCLoop";
  Base {
    ccmodels = "EffMassQuantum:carrier";
    poisson { ... }
    carrier {
      schroedinger_c_0 { ... }
      schroedinger_c_1 { ... }
      schroedinger_c_2 { ... }
      schroedinger_v_0 { ... }
      schroedinger { ... }
} } }
```

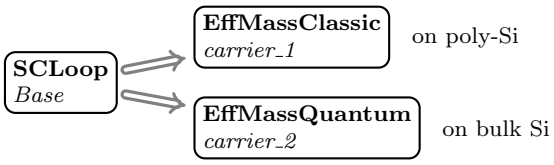
Fig. 4 Same as Fig. 3 but with an equilibrium distribution of confined carrier states; the carrier model instance invokes one closed-boundary Schrödinger model for every conduction/valence valley (c_0, c_1, ..., v_0, ...); The model instances can be configured via their respective sections (schroedinger_c_0, ...) or the common *schroedinger* section.

the base (real-space) device; additional devices (also k-space) can be specified in their respective sections. `Materials` contains a nested section for each material known to VSP. A *material database* (see Section 3.5) is included with VSP and contains parameter values for common semiconductors (Si, Ge, GaAs, ...) and insulators (SiO_2, HfO_2, ...) as well as metals. The `Simulation` section contains all the data and control flow information of the simulation work flow.

3.3 Simulation Flow Control

In the `Simulation` section both the control flow and the data flow can be specified. Each model instance is configured via a section with submodels corresponding to nested sections. A *base model* is specified that is instantiated at the start of the simulation. Further model instances are necessarily submodels of the *base model*.

The example in Fig. 3 illustrates this: The *base model* is **SCLoop** and its instance is referred to as *Base* and configured by a nested section of the same name. The **SCLoop** instantiates two submodels, *poisson* and *carrier*; the *poisson* instance is "hard-wired" into the model **SCLoop**, whereas the carrier model class and instance name are chosen by the user. Any model can




```
Simulation {
  model = "SCLoop";
  Base {
    ccmodels = ["EffMassClassic:carrier_c",
                "EffMassQuantum:carrier_q"];
    carrier_c_segments = "PolyGate";
    carrier_q_segments = "Bulk";
    carrier_c { ... }
    carrier_q { ... }
} }
```

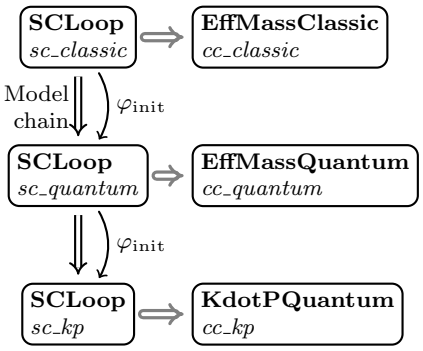
Fig. 5 Different carrier models can be used on different segments, where appropriate; here, a classic carrier distribution is sufficient to model the accumulation/depletion effects in the poly-Si gate, while quantum confinement is accounted for in the channel.

be used as a carrier model, as long as it provides a suitable interface, i.e. an input quantity `phi`/φ and output quantities `ccn`/n, `ccp`/p, `dccn`/dn, and `dccp`/dp. Since the interface is predefined, the **SCLoop** instance takes care of transferring the quantities.

In Fig. 3 we have used a model providing a classical carrier distribution; in Fig. 4 we have exchanged the carrier model to a more sophisticated one including carrier confinement. The general procedure is the same as before, only the carrier model has submodels of its own, i.e. instances of the **Schroedinger** model that calculate the bound states for each conduction band valley and valence band (`c_0`, `c_1`, `c_2`, `v_0`, ...). The interface to the **Schroedinger** instance is predefined: it takes the potential `V`/V and effective mass `inv_mass`/$(\underset{\sim}{m}^*)^{-1}$ as input quantities, and returns the energies `E`/E_n and wavefunctions `psi`/ψ_n. Multiple carrier models can be used, each run on a different portion of the simulation domain (cf. Fig. 5). Different models can be instantiated and their instances configured separately.

3.3.1 Stepping and Logging

Stepping lets VSP run multiple simulations with one or more attributes being stepped through a list of values, while *logging* writes specified data to a file for each step. A basic application of this feature is the calculation of characteristics, such as capacitance-voltage (C/V) and current-voltage (I/V). *Stepping* and *logging* are demonstrated in the following example where charge is computed as function of temperature:




```
Simulation {
  model = "Chain";
  Base {
    models = ["SCLoop:sc_classic",
              "SCLoop:sc_quantum",
              "SCLoop:sc_kp"];

    sc_classic {
      ccd = "~Device.AcceptorConcentration";
                ... }
    sc_quantum { phi = "^sc_classic.phi";
                ... }
    sc_kp      { phi = "^sc_quantum.phi";
                ... }
} }
```

Fig. 6 A *chain* instantiates a list of models and invokes each of them in sequence when run; *attributes* can be passed between the model instances.

```
Simulation {
  model = "SCLoop";
  Base {
    T = step([77 K, 120 K, 200 K,
              250 K, 300 K], pri = 10);
  }

  Logging {
    file = "cv-curve";
    variables = [
      "~Simulation.Base.T",
      "~Simulation.Base.Gate.charge"];
} }
```

3.3.2 Chains

The **Chain** is an abstract model that is used to run a sequence of submodels. *Chains* can be nested and give the user complete control over model execution. Figure 6 demonstrates the usage of a *chain*; here the *base model* is a **Chain** and instantiates the three **SCLoop** submodels, `sc_classic`, `sc_quantum`, and `sc_kp`. Also shown is the passing of attributes between model instances which will be discussed in the next section.

3.4 Data Flow Control

Data flow between models is either intrinsic or user-defined. The former can be found in the examples in Figs. 3 to 5; here the attributes that are exchanged between the **SCLoop** model and the carrier model are part of the predefined interface. In Fig. 6 we can see an example of user-defined data flow; Through the setting `phi = "^sc_classic.phi"` in the `sc_quantum` section, we instruct the model instance to fetch the quantity `phi` from instance *sc_classic* and store it in its own quantity `phi`. Such assignments allow data to be transferred between any two model instances in the configuration tree. The same way data can be transferred from the input device: `cca = "~Device.AcceptorConcentration"` gets the quantity `AcceptorConcentration` that has been read with the input geometry and writes it to `cca`.

Any model instance can have its data redirected to a file for visualization or debugging. Every model configuration section can have a `WriteQuans` and `WriteParams` section nested within; the former is for *quantities* only, the latter for *properties* and *parameters*.

```
WriteQuans {
    file = "some/output/file";
    variables = ["phi", "ccn", "ccp"];
    format = ["dev", "vtk", "crv"];
}
WriteParams {
    file = "other/output/file";
    variables = "*";
}
```

The `variables` parameter is a list of attributes to be written to a specified file, setting `variables = "*"` causes output of all attributes suitable for output. The path definition is platform independent. The file format can be specified for quantity output; supported formats are the proprietary GTS DEV format, the VTK XML format for unstructured grid data [17], or a text file (CRV format, 1D only).

3.5 Material Database

The top-level `Materials` section in the IPD contains information for each material supported by VSP and is referred to as *material database*. `Materials` contains one subsection for each material, e.g. `Materials.Si`, or material combination, e.g. `Materials.GaAsSb`. Each material subsection contains one subsection for each material property such as `MaterialTypeModel`, `PoissonModel`, `ElasticityModel`, `EffectiveMassModel`, and `KPModel`. Each of the property sections is read by the corresponding *material model*, which provide a standardized interface for material parameters and serve the information suitably to VSP's models as C++ data structures.

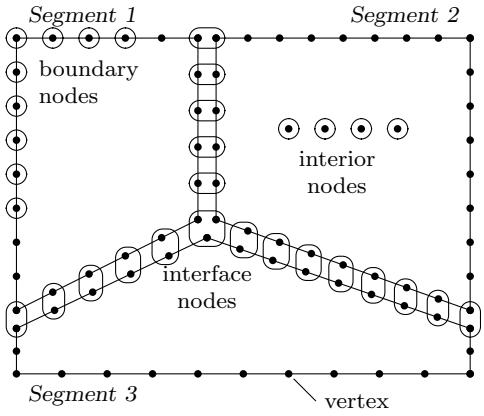


Fig. 7 The role of *nodes* in VSP becomes clear when looking at devices with more than one *segment*. Every *segment* contains its own set of *vertices*, rather than sharing one set of *vertices* for the whole domain. At the interface between two *segments* we may have two (or more) *vertices* representing the same point in space. *Nodes* resolve this ambiguity by referencing the interface nodes.

4 Software Architecture – Levels of Abstraction

VSP is a monolithic simulation program written in C++. The VSP software architecture is divided in three levels of abstraction: (i) the data and topology level, (ii) the modeling level, and (iii) the algebraic level. While the first two are dependent on each other, the third is independent to allow a maximum of *flexibility*.

4.1 Data and Topology Level

Data and topology form the foundation of numerical modeling and simulation. This level can be regarded as the *low level* of VSP's infrastructures, whereas the modeling level to be described in the next section would be the *high level*.

The simulation domains of VSP (in real space or k-space) are called *devices*. A *device* is organized in *segments*. *Segments* are filled with *elements* which are simplices spanned between their *vertices* – also contained in the respective *segment*. Each *segment* is independent, i.e. no *elements* or *vertices* are shared between *segments*. Global, i.e. *device*-wide connectivity is provided by *nodes*. Figure 7 shows a sketch of a VSP device, highlighting the role of *nodes*. Figure 8 displays the topological relations between *segment*, *element*, *vertex*, and *node*. Note that most relations are bidirectional, allowing to go from any topological entity to any other by following the references; some additional references that serve as shortcuts (e.g. Element → Node) have been omitted for clarity.

VSP can store structured data on all of the aforementioned topological entities, except on *nodes*, in the

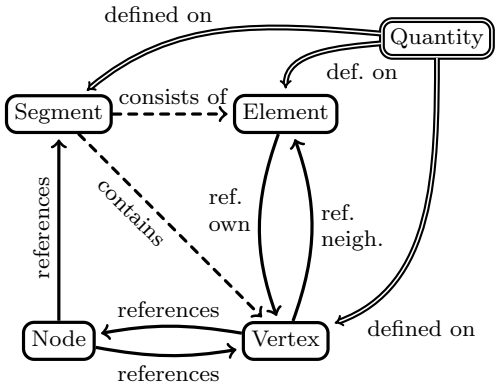


Fig. 8 Topological relations between objects in VSP; *segments* contain both *elements* and *vertices*. *Vertices* contain their location in real/k-space, while *elements* contain geometrical data for computing couplings; *elements* and *vertices* reference each other. *Nodes* reference one or more *vertices* along with their corresponding *segments*. *Quantities* can be defined on *vertices*, *elements*, or *segments*.

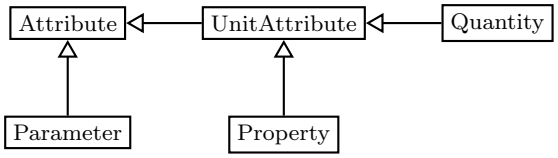


Fig. 9 Relations between attribute types in VSP; the base type for data storage and exchange is the *Attribute*; *Parameter* is a direct descendant of *Attribute*; *UnitAttribute* stores a physical unit along with the raw data; *Property* and *Quantity* are its descendants.

form of *quantities*. *Quantities* are one form of model *attributes*, the other two being *parameters* and *properties* – discussed already in Section 3.1. The relation between different kinds of *attributes* is shown in Fig. 9. All *attributes* of a model are available for data exchange; the user can instruct VSP to transfer *attributes* between model instances, demonstrated in Section 3.3.2.

Quantities may store any kind of position-dependent or k-dependent data; the data may be be scalar (`double`, `complex16`, `int`), vector-valued (`Tuple<T>`), or tensor-valued (`Transform<T>`). A *quantity* may be represented in arrays of any number of dimensions: zero-dimensional (e.g. potential, carrier concentration), one-dimensional (e.g. single-band wavefunctions), two-dimensional (e.g. multi-band wavefunctions), four-dimensional (e.g. multi-band wavefunctions, k-resolved), and so on.

On construction, every *attribute* must be provided with a data type, an identifier, a brief description, and a tag indicating the *attribute's* usage such as input, output, and internal. *Properties* and *quantities* must also be provided with a physical unit. The given information (type, identifier, description, tag, and unit) is used to refer to the *attribute*, verify data flow, and as part of the automated model documentation generation referred to as *literate-modeling* (cf. Section 4.4).

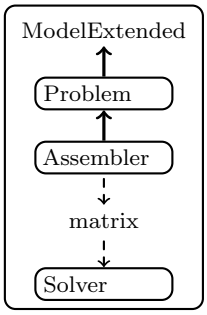


Fig. 10 Typical algorithmic layout of a VSP model; a `Problem` instance is generated that uses the topological structure provided by the `ModelExtended` instance. The topology of boundary conditions is handled by the `Problem` instance. An `Assembler` instance uses the information provided by the `Problem` instance along with geometrical information from the model to discretize the equation and assemble a matrix. The matrix is processed by a solver instance.

4.2 Modeling Level

The previous section introduced terms such as *topology*, *data*, and *model attributes*. On the modeling level we are concerned with higher-level items, such as *mathematical expression*, *(differential/integral) equations*, or *boundary conditions*. VSP provides a high-level interface to deal with these items – the `ModelExtended` class.

A model derived from `ModelExtended` inherits all the required infrastructure required for translation between the modeling and the topology or data levels. A typical model layout is shown in Fig. 10. It shows two additional modules: `Problem` and `Assembler`; they are required when partial differential equations need to be solved numerically, but are optional otherwise. `Problem` takes topological information about the boundary conditions, i.e. where and of what type they are. The information is processed using the low-level topological information provided by `ModelExtended` to pre-determine the size and structure of the discrete equation system. The boundary conditions are passed to the `Assembler` instance in equation form, along with the equations for the interior points of the domain. The `Assembler` instance can then generate a system matrix, which may be passed to a numerical solver.

VSP allows the use of symbolic math for specifying equations, expression evaluation, integration, and similar tasks. The following code lines serve as illustration:

```
// vector potential
A   = 0.5 * B0 * cross(ez, position);

// density from wavefunction
rho = magsq(psi);

// electric field
E   = -grad(phi);

// calculating the centroid
Tuple<> center =
   integrate(position) / integrate(1.0);
```

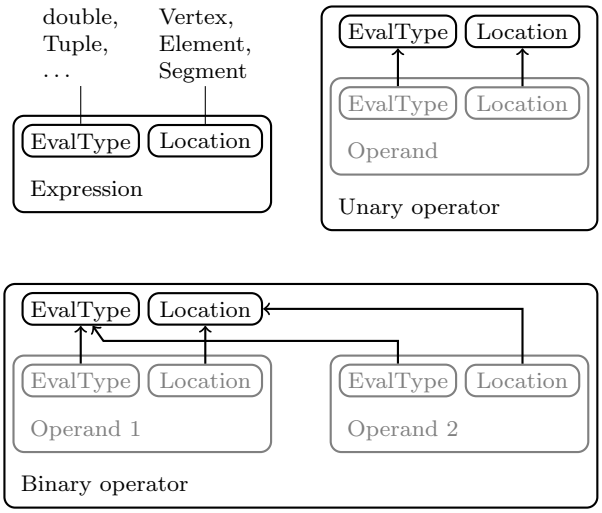


Fig. 11 Expression typing scheme in VSP; each expression contains its evaluation type (`double`, `complex16`, `Tuple`, `Tensor`, ...) and location tag (`Vertex`, `Element`, `Segment`) as nested types. Unary and binary operations derive their evaluation type and location based on the operands' evaluation and location types.

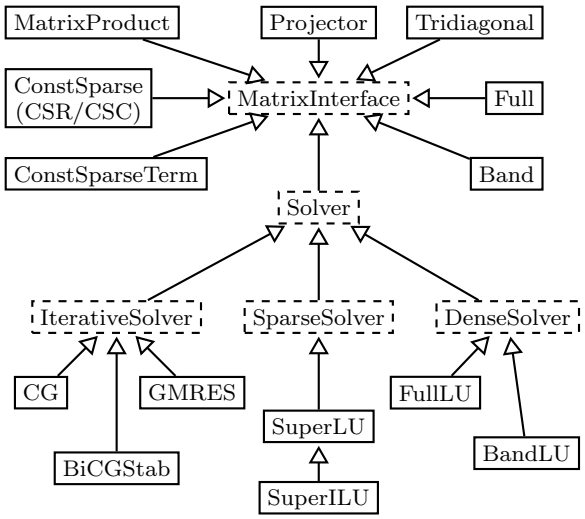


Fig. 12 Relation map of the algebraic operators in VSP; the base type is `MatrixInterface`. The common interface allows to represent dense or sparse matrices, projections, solvers (direct, sparse direct, iterative), and combinations of these as generic algebraic operators.

Every combination of terms has its own C++ type as shown in Fig. 11; nested within are the type to which the expression evaluates, `EvalType`, and a tag to represent the location of evaluation, `Location`. Operators or functions applied to expressions are aware of both evaluation and location type and may modify them according to specific rules. For instance `magsq(...)`, the square-magnitude function will change the `EvalType` from `double`, `complex16`, or `Tuple<T>` to `double`, while `grad(...)` will change `EvalType` to `Tuple<EvalType>` and `Location` from `Vertex` to `Element`. The static typing system in C++ serves as a formal correctness check for all expressions.

4.3 Algebraic Level

The *algebraic level* is detached from the low-level picture of the topological and data level and the high-level picture of the modeling level. It provides abstraction of entities such as matrices, solvers, and projections to a generic finite-dimensional linear operator, called `MatrixInterface`. A `MatrixInterface` object has the property of dimension and provides various methods for multiplication by a vector (or multiple vectors) from left and right, as well as evaluation of bilinear forms. Derivate classes of `MatrixInterface` are required to at least implement left and right multiplication as a minimal set of operations. The remaining methods can be constructed by `MatrixInterface` automatically. Figure 12 shows all the algebraic operator classes and their relation to `MatrixInterface`.

4.4 Literate Modeling

In [7] the authors of NEMO5 point out: "Being a research code employed by changing generations of students, documentation, clarity, and modularity of the code are essential. Only when all these criteria are fulfilled, can junior researchers act as builders of individual modules and the code endure multiple generations of developers."

In VSP we go even further by introducing the notion of *literate modeling*. It borrows from the concept of literate programming by D. Knuth [18] in which the program and its description are written as one document from which code and documentation can be extracted. The VSP code provides facilities to embed documentation into the models themselves. Their structure is based on *topic-oriented authoring* [19]. The following code snippet serves as illustration:

```
struct Schroedinger : ModelExtended {
    ...
};

EXTERN_DOCUMENT(Topic, models_unstr)
DECLARE_DOCUMENT(
    ModelNode<vsp::Schroedinger>,
    Schroedinger)
DOCUMENT(Schroedinger,
    topic = &models_unstr,
    description = "This␣model␣solves␣the␣"
        "closed␣boundary␣single␣band␣"
        "Schroedinger␣equation␣...")
```

The example only shows how to add a brief description to a model but the model description can be structured into several paragraphs. The documentation is organized in nodes representing sections and subsections of the

documentation. The node for the `Schroedinger` model is added to the `Topic` node `models_unstr` which represents the section containing descriptions of all VSP models operating on unstructured grids. The information provided with the model's *attributes* (cf. Section 4.1) is automatically compiled into its documentation node. Also, every of the important IPD sections (`Device`, `Simulation`, `Logging`, `WriteQuans`, `WriteParams`) are documented in this manner.

The documentation is contained within the VSP binary and can be accessed by the user. The user can obtain formatted output for any documentation node by running VSP in documentation mode. This is useful as a quick reference for models and IPD sections and lowers the learning barrier of VSP especially for new users. Another usage is the output of the entire documentation that can then be compiled into a manual using a document preparation system such as LaTeX. Additionally, the documentation system features automated generation of IPD defaults that can be included in simulation IPDs.

5 Physical Models

In this section we present some of the physical models available in the VSP. A concise overview of the models capabilities and properties is given.

5.1 Self-Consistent Loop

The self-consistent loop model **SCLoop** is used to iterate between the Poisson model and the carrier models until the electrostatic potential converges. The self-consistent loop model takes care of setting the correct *parameters*, *properties*, and *quantities* of the submodels and getting the calculated *quantities* to update the potential and the space charge density, respectively. The potential is updated according to

$$\boldsymbol{\varphi}_n = \boldsymbol{\varphi}_{n-1} + d\delta\boldsymbol{\varphi}_n, \tag{1}$$

where d is the damping by the parameter $d \in [0, 1]$. Different schemes are available for computing d: (i) the `simple` damping scheme, where the parameter d is constant, (ii) the `mmnt` (or alternatively `potential`) scheme, ported from the device simulator MINIMOS-NT [20,21], which takes the exponential relation between potential and carrier concentration into account and (iii) the `cosine` scheme defined by

$$d = \min\left(1.0, 1.0 + \frac{\langle \delta\boldsymbol{\varphi}_n, \delta\boldsymbol{\varphi}_{n-1}\rangle}{\|\delta\boldsymbol{\varphi}_n\|_2 \left\|\delta\boldsymbol{\varphi}_{n-1}\right\|_2}\right), \tag{2}$$

which prevents oscillatory updates. In all cases, the range of d can be restricted to a user-specified interval $[$`dmin`, `dmax`$]$. The system is considered convergent if the calculated potential update norm drops below a given threshold value (`tolerance`).

The **SCLoop** model also defines the settings of the contact regions through the **Contact** model. In the **Contact** model the `Voltage` or `Fermi` (i.e. Dirichlet or Neumann) boundary conditions are set to the user-defined values. The model also includes the work function difference for metals. A *contact* also defines the region where space charge is summed up and thereby allows the calculation of C/V characteristics.

5.2 Poisson Model

Self-consistent simulations necessitate the solution of the Poisson equation

$$\boldsymbol{\nabla} \cdot \left(\underset{\sim}{\epsilon}\boldsymbol{\nabla}\varphi\right) + \varrho = 0, \tag{3}$$

where φ is the electrostatic potential, $\underset{\sim}{\epsilon}$ is the electric permittivity tensor, and ϱ is the free space charge. In semiconductors charge consists of ionized dopants of acceptor (N_A) and donor type (N_D), and of electrons and holes.

$$\varrho(\varphi) = -q_0\big(n(\varphi) - p(\varphi) + N_\mathrm{A} - N_\mathrm{D}\big) \tag{4}$$

Charge itself depends on the electrostatic potential since electron and hole concentrations are influenced by the band edge energy and the Fermi level producing an non-linear Poisson equation.

For a stable iteration towards self-consistency the Poisson equation needs to be linearized to give

$$\boldsymbol{\nabla} \cdot \left[\underset{\sim}{\epsilon}\boldsymbol{\nabla}(\varphi_0 + \delta\varphi)\right] + \varrho + \frac{\mathrm{d}\varrho}{\mathrm{d}\varphi}\delta\varphi = 0. \tag{5}$$

The derivative can be expressed analytically only for a 3D carrier gas, while approximations can be found for confined carriers [22,23].

5.3 Carrier Models

Carrier models take the electrostatic potential and the (quasi-)Fermi level as input and calculate the carrier concentration and its derivative for a given geometry and material configuration. Any model fulfilling the basic interface depicted in Figs. 3 and 4 can be used as a carrier model within a self-consistent loop (by the **SCLoop** model). VSP allows the combination of different carrier models within a self-consistent loop, for example classically calculated carriers in the contact regions and quantum-mechanically confined carriers in the

Table 1 The supply function SF, the effective density of states N_C and the density of states mass m_dos are summarized for carrier gases of dimension d.

d	SF^d	N_C^d	m_dos^d
0D	$\frac{1}{1+\exp\left(\frac{E_s-E_\mathrm{F}}{k_\mathrm{B}T}\right)}$	g	1
1D	$\mathcal{F}_{-\frac{1}{2}}\left(-\frac{E_s-E_\mathrm{F}}{k_\mathrm{B}T}\right)$	$g\left(\frac{m_\mathrm{dos}^\mathrm{1D}k_\mathrm{B}T}{2\pi\hbar^2}\right)^{\frac{1}{2}}$	$m_\parallel$
2D	$\ln\left(1+\mathrm{e}^{\frac{E_\mathrm{F}-E_s}{k_\mathrm{B}T}}\right)$	$g\frac{m_\mathrm{dos}^\mathrm{2D}k_\mathrm{B}T}{2\pi\hbar^2}$	$\sqrt{\det(\underset{\sim}{m}_\parallel)}$
3D	$\mathcal{F}_{\frac{1}{2}}\left(-\frac{E_\mathrm{c}-E_\mathrm{F}}{k_\mathrm{B}T}\right)$	$g\left(\frac{m_\mathrm{dos}^\mathrm{3D}k_\mathrm{B}T}{2\pi\hbar^2}\right)^{\frac{3}{2}}$	$\sqrt[3]{\det(\underset{\sim}{m})}$

channel region. The currently available carrier models are outlined in the following sections.

5.3.1 Classical Carrier Model

The classical carrier model (**EffMassClassic**) assumes that the electrons and holes behave like a free carrier gas. Assuming a Fermi distribution of the carriers the total concentration can be expressed in terms of the effective density of states N_C^3D and the complete Fermi integral $\mathcal{F}_{\frac{1}{2}}$. The effective density of states also includes degeneracy $g = 2g_\mathrm{V}$ due to spin and equivalent valleys g_V. It depends on the density of states mass $m_\mathrm{dos}^\mathrm{3D}$. A summary is given in Table 1.

The classical carrier model extracts the necessary material parameters from the *material database* and sums up the contributions to the total carrier concentration from all relevant valleys.

5.3.2 Parabolic EMA Quantum

For confined systems we provide the carrier model **EffMassQuantum** which involves solving the single-band Schrödinger equation with constant effective mass. Geometric confinement can be in one, two, and three dimensions which translates to the treatment of two-, one-, and zero-dimensional carrier gases, respectively. The time-independent m-dimensional single-band effective mass Schrödinger equation is solved with closed (i.e. Dirichlet) boundary conditions at the quantum domain boundaries. For that purpose **EffMassQuantum** instantiates and invokes **Schroedinger** submodels.

Carrier concentration is obtained by summation/integration of states up to a certain *truncation energy* E_trunc which is set to several $k_\mathrm{B}T$ above the Fermi level E_F. After calculating all the confined states with eigenenergy E_s below E_trunc the total carrier concentration

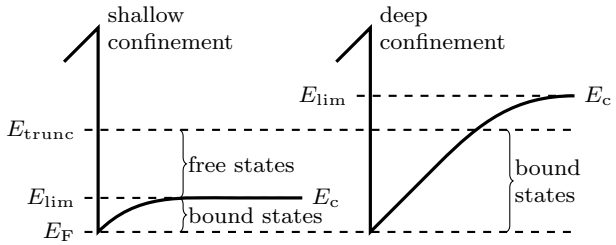


Fig. 13 Shallow (left) and deep (right) confinement regime; in the former mobile charge is comprised of bound carriers with energies up to E_lim and free carrier above E_lim modeled as classic 3D carrier gas. Shallow confinement is typical of carrier accumulation. In deep confinement E_lim lies above truncation energy E_trunc and charge only consists of bound carriers. Deep confinement is typical of inversion and fully-depleted channels.

is obtained by

$$n(\mathbf{r}) = \sum_v \sum_s \left|\psi_{vs}^{m\mathrm{D}}(\mathbf{r})\right|^2 N_{\mathrm{C},vs}^{3-m\mathrm{D}} \mathrm{SF}_{vs}^{3-m\mathrm{D}}. \tag{6}$$

In some cases not all the states between the band edge and E_trunc are confined as outlined in Fig. 13. In the case of *shallow confinement* confined carriers are obtained only bound up to E_lim and the remainder is treated as free carrier gas

$$\begin{aligned} n(\mathbf{r}) = &\sum_v \sum_s \left|\psi_{vs}^{m\mathrm{D}}(\mathbf{r})\right|^2 N_{\mathrm{C},vs}^{3-m\mathrm{D}} \mathrm{SF}_{vs}^{3-m\mathrm{D}} \\ &+ N_\mathrm{C}^\mathrm{3D} \mathcal{F}_{\frac{1}{2}}\left(-\tfrac{E_\mathrm{c}-E_\mathrm{F}}{k_\mathrm{B}T}, -\tfrac{E_\mathrm{lim}-E_\mathrm{F}}{k_\mathrm{B}T}\right), \end{aligned} \tag{7}$$

making use of the incomplete Fermi integral for the free carrier concentration.

The transition energy between bound and unbound states, E_lim is automatically determined by the model based on the device topology. As sketched in Fig. 14 the minimum of the band edge is computed across the bounding segments that constitute *open boundaries* and assigned to E_lim. Dielectrics are considered *barriers* and are omitted in the E_lim calculation, hence carriers can become confined next to dielectric segments.

The $d = 3 - m$ dimensional effective densities of states N_C^d and corresponding supply functions are summarized in Table 1. The summation is carried out over all subbands s and valleys v. If bound states penetrate into barriers, a weighted average of the mass is used in Eq. (6), $\underset{\sim}{m}_s^{-1} = \langle \underset{\sim}{m}^{-1} \rangle_s$. In a 1D carrier gas we project $\underset{\sim}{m}_s^{-1}$ along the axis of free movement $\mathbf{e}_\parallel$ to obtain the transport and density-of-states effective mass $m_\parallel = \mathbf{e}_\parallel \cdot \underset{\sim}{m}_s \cdot \mathbf{e}_\parallel$. In a 2D carrier gas the transport effective mass is a tensor. The density-of-states mass is obtained from $\sqrt{\det(\underset{\sim}{m}_\parallel)}$, with

$$\underset{\sim}{m}_\parallel = \underset{\sim}{\tau}_\parallel \cdot \underset{\sim}{m}_s \cdot \underset{\sim}{\tau}_\parallel + \underset{\sim}{\iota} - \underset{\sim}{\tau}_\parallel, \tag{8}$$

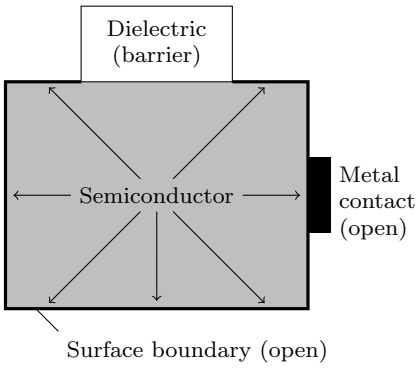


Fig. 14 The transition energy E_{lim} between bound and free states is determined by looking at the problem topology. The segments surrounding the region where carrier concentration need to be computed, i.e. semiconductor segments, are broken down into barriers and open boundaries. Since carriers can escape the region via open boundaries only, E_{lim} is computed as the potential minimum along the open boundary segments.

$\underset{\sim}{\tau}_{\parallel}$ and $\underset{\sim}{\iota}$ being the projection onto the free movement axes and the identity tensor, respectively. This way the model can deal with truly arbitrary crystal orientations.

5.3.3 Parabolic EMA NEGF

For nanoscaled devices, numerical simulations based on the non-equilibrium Green's function (NEGF) formalism are commonly performed [24–27]. A very efficient implementation of this method has been achieved by means of a recursive algorithm [28]. Proper numerical integration methods are vital for the stability and accuracy of NEGF simulations.

The retarded and advanced Green's functions are determined by the equation

$$\begin{aligned} G^{\mathrm{R}}(\mathbf{r},\mathbf{r'},E) &= G^{\mathrm{A}\dagger}(\mathbf{r},\mathbf{r'},E) \\ &= [EI - H(\mathbf{r},\mathbf{r'},E) - \Sigma^{\mathrm{R}}(\mathbf{r},\mathbf{r'},E)]^{-1}, \end{aligned} \tag{9}$$

where $H(\mathbf{r},\mathbf{r'},E)$ is the Hamiltonian of the system and $\Sigma^{\mathrm{R}}(\mathbf{r},\mathbf{r'},E)$ is the retarded self-energy. The lesser Green's function is calculated as

$$G^{<}(\mathbf{r},\mathbf{r'},E) = G^{\mathrm{R}}(\mathbf{r},\mathbf{r'},E)\Sigma^{<}(\mathbf{r},\mathbf{r'},E)G^{\mathrm{A}}(\mathbf{r},\mathbf{r'},E).$$

The lesser self energy of the left and right contact is given by $\Sigma^{<}_{\mathrm{l,r}}(E) = \mathrm{i}\Im\left\{\Sigma^{\mathrm{R}}_{\mathrm{l,r}}(E)\right\} f_{\mathrm{l,r}}(E)$ with the occupation function $f_{\mathrm{l,r}}(E)$ of the left and right lead, respectively. The Green's functions allow the calculation of physical quantities of interest such as the local density of states, $N(\mathbf{r},\mathbf{r},E) = -\frac{1}{\pi}\Im\left\{G^{\mathrm{R}}(\mathbf{r},\mathbf{r},E)\right\}$, and the electron and current density

$$n(\mathbf{r}) = -2\mathrm{i}\int G^{<}(\mathbf{r},\mathbf{r},E)\frac{dE}{2\pi}, \tag{10}$$

$$j(\mathbf{r}) = -\frac{\hbar q}{m^*}\int \left[(\nabla - \nabla')G^{<}(\mathbf{r},\mathbf{r'},E)\right]\Big|_{\mathbf{r'}=\mathbf{r}}\frac{dE}{2\pi}. \tag{11}$$

The numerical evaluation of these quantities requires a discretization of the energy space. A simple approach using an equidistant energy grid suffers from two problems. A small number of grid points will not correctly resolve narrow resonances, whereas a vast number can lead to an unpredictable summation of numerical errors and to intractable memory requirements. These effects can lead to instability or poor convergence of a self-consistent iteration loop [29]. Adaptive energy integration (AEI) on a non-equidistant grid is required to increase accuracy, numerical stability, and memory efficiency. Section 6.4 outlines the different approaches that were implemented and tested for applicability to the NEGF formalism [30].

The NEGF carrier models (**EffMassNEGF**) are currently available for 1D and 2D orthogonal geometries in the ballistic regime. In the 2D case, we implemented the Svizhenko algorithm [28]. For 1D simulations we additionally provide the algorithm presented by Lake [24] which models equilibrium contact regions.

5.3.4 **k·p** *Quantum*

The **k·p** method is a popular method for describing band structure in the vicinity of high-symmetry points. The resulting effective Hamiltonians consists of matrices, or systems of coupled Schrödinger equations, in which states are expressed as envelope functions of each band's basis states [31]. In that sense, the single-band effective mass Schrödinger equation itself can be seen as special case of the multi-band **k·p** theory, where all interactions with the other (remote) bands are described by a single parameter, namely the effective mass.

In VSP the **k·p** method is implemented within the **KdotPQuantum** model. The model behaves analogously to the **EffMassQuantum** model described in Section 5.3.2 and they share the same IPD interface. Instead of using analytical formulae for carrier population **KdotPQuantum** performs a numerical k-space integration of the **k·p** (sub-)band structure. The eigenenergies and wavefunctions are obtained for each k-grid point by invoking instances of the **SchroedingerMulti** model. The consideration regarding bound and unbound states mentioned in Section 5.3.2 also applies here. To save computational resources the **KdotPQuantum** model applies a heuristic search algorithm to efficiently determine which k-grid points contribute significantly to the carrier concentration and omits those that don't.

VSP's **k·p** capabilities have been used for investigating properties of a variety of nanoelectronic devices: p-type silicon ultra-thin-body (UTB) channels [32] and quantum dots [32] using the six-band Luttinger-Kohn model [33], n-type silicon UTB [34] and nanowire chan-

nels [35,36] using the two-band model in ref. [37], quantum cascade lasers devices [38] using two-band [39] as well as four- and eight-band models for III-V materials [40], or lead-salt based nanostructures using the four-band Hamiltonian from ref. [41].

VSP provides the generic `KPModel` interface in the *material database* that allows users to specify multi-band Hamiltonians. A general model is used for the diagonal and coupling **k**·**p**-Hamiltonian components up to second order in **k**:

$$\mathbf{H} = \begin{pmatrix} H_{11} & H_{12} \cdots \\ & H_{22} \cdots \\ \text{c.c.} & \ddots \end{pmatrix}, \qquad \begin{aligned} H_{lm} = \frac{\hbar^2}{2}\mathbf{k} \cdot \underset{\sim}{m}_{lm}^{-1} \cdot \mathbf{k} \\ + \hbar \mathbf{v}_{lm} \cdot \mathbf{k} + U_{lm} \end{aligned} \tag{12}$$

The second order contributions are controlled by the inverse (coupling) effective mass tensor $\underset{\sim}{m}_{lm}$, the first order contributions by the (coupling) group velocities $\mathbf{v}_{lm}$, and the zero-order contributions through the (coupling) potentials U_{lm}. One way to specify the Hamiltonian elements H_{lm} is via a Cartesian coordinate system (`symmetry = "G"`), with $\mathbf{e}_x, \mathbf{e}_y, \mathbf{e}_z$ as base; $\underset{\sim}{m}_{lm}$ and $\mathbf{v}_{lm}$ are specified via `mXX`, `mYY`, `mXY`, ..., and `vX`, `vY`, and `vZ`, respectively. In case the **k**·**p**-Hamiltonian expands around one of the boundary points X or L (`symmetry = "X"` or `"L"`), the coefficient may be defined using a basis of longitudinal and transversal directions $\mathbf{e}_\mathrm{l}, \mathbf{e}_\mathrm{t}, \mathbf{e}_{\mathrm{t}'}$ and `ml`/`mt` and `mlt`/`mtt` are used for diagonal and off-diagonal elements of $\underset{\sim}{m}_{lm}$, and `vl` and `vt` for $\mathbf{v}_{lm}$. VSP replicates and rotates the Hamiltonian for every occurrence of the same symmetry point, i.e. [100], [010], and [001] for X and [111], [$\bar{1}$11], [1$\bar{1}$1], and [11$\bar{1}$] for L.

Strain is included via deformation potentials D_{lm}^{ij} for each strain component. The resulting potentials are added to U_{lm},

$$U_{lm} \mapsto U_{lm} + \varepsilon^{ij} D_{lm}^{ij}. \tag{13}$$

Deformation potentials D_{lm}^{ij} are specified using the same notation as the components of $\underset{\sim}{m}_{lm}$.

Any conceivable **k**·**p**-Hamiltonian that fits the representation in Eq. (12) may be be specified and multiple models for different symmetry points and bands may be combined. As an example we give the complete specification of the two-band **k**·**p** Hamiltonian for the silicon conduction band from ref. [37].

```
TwoBandConduction {
  symmetry = "X";
  degeneracy = 2; // spin

  a = 5.431 "Angstrom";
  Eg = 1.12 "eV";

  H = [["H1", "HC"],
       ["HC", "H2"]];

  H1 { type = "conduction";
       ml = 0.916;
       mt = 0.196;
       vl = 0.15 * 2.0 * pi / ^a / ml;
       U = 0.5 * ^Eg;
       Dl = 9.0 "eV"; }

  H2 : H1 { vl = -^H1.vl; }

  HC { type = "coupling";
       inv_mtt = 2.0 * (1.0 / ^H1.mt - 1.0);
       Dtt = 7.0 "eV"; }
}
```

The general structure of the 2×2 Hamiltonian is specified in the variable `H`. Each of the referenced components `H1`, `H2`, and `HC` is defined in its individual section below using expressions for masses, velocities, and (deformation) potentials.

5.4 Pauli Master Equation

Theoretical studies have shown that in many practical cases the steady state transport in quantum cascade lasers (QCLs) is incoherent and a semi-classical description was found to be sufficient [42,43]. Following this approach, we developed a transport model for QCLs based on the Pauli master equation (PME) [44]. Transport is described via in and out-scattering between quasi-stationary basis states, which are found by solving the Schrödinger equation. The Hamiltonian includes the band edge formed by the heterostructure, thus, tunneling is accounted for through the delocalized eigenstates. Transport occurs via scattering between these states.

The transport equations are derived from the Liouville-von Neumann equation in the Markov limit in combination with the diagonal approximation. This means that off-diagonal elements of the density matrix are neglected and one arrives at the Boltzmann-like PME [45].

$$\frac{\mathrm{d}f_{\mathbf{k},n}(t)}{\mathrm{d}t} = \sum_{\mathbf{k}',m} \left\{ \Gamma_{m,n}\left(\mathbf{k}',\mathbf{k}\right) f_{\mathbf{k}',m}\left(t\right) - \Gamma_{n,m}\left(\mathbf{k},\mathbf{k}'\right) f_{\mathbf{k},n}\left(t\right) \right\} \tag{14}$$

The transition rate from state $|\mathbf{k}, n\rangle$ to state $|\mathbf{k}', m\rangle$ for an interaction H_int follows from Fermi's golden rule

$$\Gamma_{n,m}\left(\mathbf{k},\mathbf{k}'\right) = \frac{2\pi}{\hbar} \left|\langle \mathbf{k}', m | H_\mathrm{int} | \mathbf{k}, n \rangle\right|^2 \delta\left(E_n(\mathbf{k}) - E_m(\mathbf{k}') \mp \hbar\omega\right). \tag{15}$$

We make use of the translational invariance of the QCL structure and simulate the electron transport over a single stage only. The wave function overlap between the central stage and spatially remote stages is small.

Therefore, the assumption that interstage scattering is limited only to the nearest neighbor stage holds and interactions between basis states of remote stages can be safely neglected. The electron states corresponding to a single stage of the QCL are determined by solving the multi-band **k**·**p** Schrödinger equation. The states of the whole QCL device structure are assumed to be a periodic repetition of the states of a central stage. This approach ensures charge conservation and allows to impose periodic boundary conditions on the PME.

Band nonparabolicity in cross-plane direction is essential to determine the subbands in QCLs. We provide a two-band **k**·**p** [39] and a four-band **k**·**p** Hamiltonian including the conduction, heavy-hole, light-hole, and split-off bands. The periodic wavefunctions are picked automatically by a reliable algorithm [38, 46].

Since transport is simulated over a central stage only, every time a carrier undergoes an interstage scattering process the electron is reinjected into the central stage with an energy changed by the voltage drop over a single period. The corresponding electron charge then contributes to the total current.

The transport equations are solved using a Monte Carlo (MC) approach. We developed an algorithm and devised several new numerical methods to reduce the computational cost of the simulation [38, 46].

The total scattering rate in a two-dimensional electron gas for a carrier in state $|\mathbf{k}, n\rangle$ for a specific scattering process is obtained by integration over all possible final wave vectors $\mathbf{k'}$ and summation of all states m

$$\Gamma_n(\mathbf{k}) = \frac{A}{(2\pi)^2} \int_{\mathrm{BZ}} \sum_m \Gamma_{n,m}(\mathbf{k}, \mathbf{k'}) \mathrm{d}\mathbf{k'} \tag{16}$$

Currently the following scattering mechanisms are included in the model:

- electron-phonon scattering due to acoustic and optical phonons modeled by a deformation potential
- polar-optical electron-phonon scattering
- alloy scattering
- intervalley scattering
- interface roughness scattering
- stimulated emission and absorption of photons

For the calculation of the rates the effect of in-plane nonparabolicity can be included using Kane's relation. For in-plane transport we developed three approaches:

1. parabolic effective (density of states) mass as input parameter
2. parabolic effective mass averaged as $\langle\psi_i|\frac{1}{m(z)}|\psi_i\rangle$ for each subband
3. non-parabolic dispersion $E(1+\alpha E) = \frac{\hbar^2 k^2}{2m}$ fitting the mass m and nonparabolicity coefficient α to the numerical subband structure determined by the Schrödinger equation

Table 2 Continuous operators and operands with their discrete counterparts

	Continuous	Discrete, element l
Gradient	$\boldsymbol{\nabla}$	$\mathbf{Z}^l = \left[-\mathbf{Y}^l[1\ 1\ \ldots]^T,\ \mathbf{Y}^l\right]$
Divergence	$\mathrm{d}V\mathrm{div}$	$\mathbf{A}^l = [\mathbf{A}^l_i,\ \mathbf{A}^l_j,\ \ldots]^T$
Control volume	$\mathrm{d}V$	$\mathbf{V}^l = \mathrm{diag}(V^l_i,\ V^l_j,\ \ldots)$
Scalar quantity q	$q(\mathbf{r})$	$\mathbf{q}^l = \mathrm{diag}(q_i,\ q_j,\ \ldots)$

Table 3 Second-order PDE terms in their discretized form

	Continuous	Discrete, element l
Laplacian	$\mathrm{d}V\nabla^2$	$\mathbf{A}^l\mathbf{Z}^l$
Anisotropic Laplacian	$\mathrm{d}V\boldsymbol{\nabla}\cdot\mathfrak{T}\cdot\boldsymbol{\nabla}$	$\mathbf{A}^l\boldsymbol{\tau}^l\mathbf{Z}^l$

6 Numerical Methods

6.1 Discretization

VSP uses a finite volume discretization scheme, thus avoiding the weak formulation fundamental to finite elements and relying instead on a formulation based on the conservation of fluxes in each of the finite volumes. Unlike to most finite volume codes, the fluxes are treated in their full vectorial form and not as projections along the edge between two points of the mesh. This is important because it is the only way material anisotropy can be introduced within a finite volume scheme. The discretization was demonstrated in [47], where the valence band states of a quantum dot were calculated using a highly anisotropic six-band **k**·**p** Hamiltonian.

Most physical laws are laws of conservation. Conservativity, therefore, serves as a common basis for the numerical modeling in our simulation framework. The finite volume method (FVM) possesses the inherent property of conservativity and is therefore well-suited as a common discretization formalism for all problems occurring in nanophysical devices.

Traditional FVM codes (Fig. 15 left) are edge-based (see e.g. [48]); a mesh node (i) couples to its neighbors (j) via the edges of the mesh graph. Each edge stores a length d_{ij} and a coupling area A_{ij}, each node stores its Voronoi cell volume V_i. The projection of the field, i.e. the derivative of a quantity φ along an edge, is approximated by $(\varphi_j - \varphi_i)/d_{ij}$. Some material property (permittivity, effective mass, ...) relates the field to a flux density which is multiplied by A_{ij} to obtain the partial flux leaving the cell. This approach has one major shortcoming: The field obtained by $(\varphi_j - \varphi_i)/d_{ij}$ is not the gradient of φ but only its projection along $\mathbf{e}_{ij}$ which implicitly assumes that the flux density caused

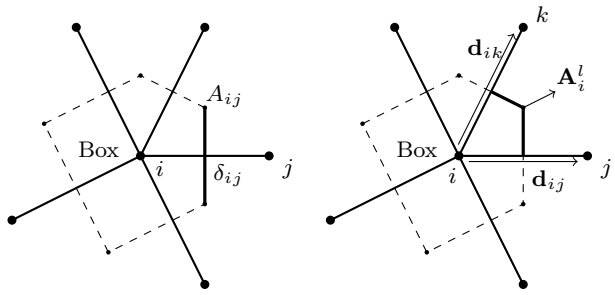


Fig. 15 Comparison of edge-based and element-based finite volume methods; in element-based FVM both gradient and coupling surface area are vector-valued entities, while in edge-based FVM they are scalar, assuming implicit projection along the edge.

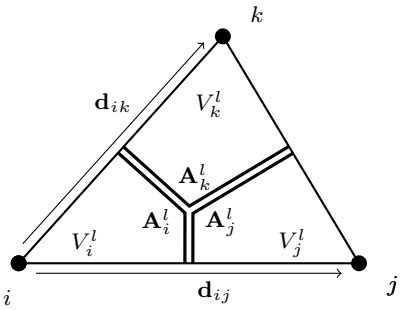


Fig. 16 Element-centric discretization; partial fluxed are evaluated according to one of the rules in Table 3. The resulting $n_\mathrm{v} \times n_\mathrm{v}$ (n_v = number of vertices per simplex) partial fluxes are added to the system matrix in their appropriate rows and columns.)

by the field is parallel to $\mathbf{e}_{ij}$ as well. This restricts the discretization to isotropic media, i.e. ones with scalar field-flux relations.

The FVM approach we use in the VSP is element-based [47]. Instead of looking at the neighbor nodes (j) of node i we look at its neighbor elements (l) as shown in Fig. 15 (right). The field is now calculated inside the element l in its vectorial form. For a simplex element (a triangle in two dimensions, a tetrahedron in three dimensions) the approximate gradient of a quantity φ is constant within the element and can be obtained by

$$[\boldsymbol{\nabla}\varphi]^l \approx \mathbf{Y}^l \begin{bmatrix} \varphi_j - \varphi_i \\ \varphi_k - \varphi_i \\ \vdots \end{bmatrix}, \quad \mathbf{Y}^l := \mathbf{U}^l \left((\mathbf{U}^l)^T \mathbf{U}^l \right)^{-1}, \quad (17)$$

where $\mathbf{U}^l := [\mathbf{d}_{ij},\ \mathbf{d}_{ik},\ \ldots]$ is a matrix containing the edge vectors of the element with respect to node i as columns. To obtain the flux density the vectorial field is multiplied with a second order tensor. The dot product of flux density and coupling area vector $\mathbf{A}_i^l$ gives the partial flux leaving the cell i via element l.

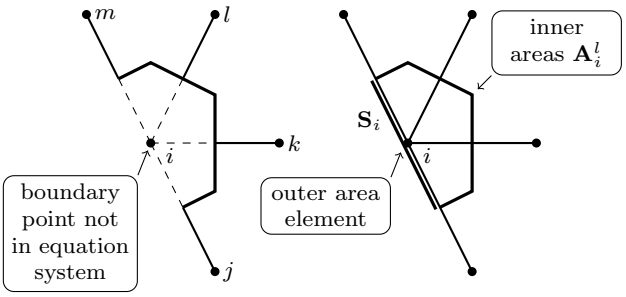


Fig. 17 Topological treatment of boundary nodes; Dirichlet nodes (left) are not represented in the system matrix but the coupling to their neighbor nodes is computed nevertheless. Neumann and Robin nodes (right) are kept in the system adding an outer self-coupling area element.

6.2 Assembly

In a 2D or 3D mesh the number of elements is several times greater than the number of nodes. To reduce the number of times a particular element has to be evaluated when building the system matrix the assembly is element-centric: A loop iterates over the elements of the simulated structure and has the partial fluxes between the element's vertices evaluated. These are then added to the appropriate elements of the system matrix.

This kind of assembly also allows the discretization of the problem's constitutive partial differential equations (PDE) to be broken down on a per-element basis. Continuous operators and operands can be directly translated into discrete ones which are represented by matrices. Table 2 shows how continuous vector-analytic operators (gradient and divergence) as well as continuous quantities are related to their discrete per-element representations as matrices. Operand matrices are diagonal and each diagonal entry corresponds to the operand's value at each of the element's vertices, hence for an n-dimensional simplex with $n_v = n + 1$ vertices, the element operands are n_v-dimensional diagonal matrices. Operators in contrast are full matrices; $\mathbf{A}^l$ is a $n_v \times 3$ matrix and contains the area vectors of the coupling surfaces between the element's vertices (see Fig. 16) as rows; Z^l is a $3 \times n_v$ matrix and relates the values at nodes to the gradient vector on the element.

Second order operators are discretized by multiplying the corresponding matrices (Table 3). This also allows the assembly of mixed derivatives such as $\partial^2/\partial x \partial y$ which are just a special case of an anisotropic Laplacian with $\underset{\sim}{\tau} = \mathbf{e}_x \otimes \mathbf{e}_y$. First order derivatives are constructed using the relation

$$\boldsymbol{\nabla} = \frac{1}{2}[\nabla^2, \mathbf{r}] = \frac{1}{2}(\nabla^2 \mathbf{r} - \mathbf{r}\nabla^2) \quad (18)$$

which is guaranteed to have an anti-symmetric discrete representation. In quantum mechanical problems even-

order operators must be symmetric and odd-order operators anti-symmetric in their discretized form.

The assembly process is automated by the `Assembler` module discussed in Section 4.2. The `Assembler` allows specification of the PDE system as a set of discrete operator equations for each *device segment*. Boundary conditions can be defined as shown in Fig. 17. The `Assembler` object can then extract a sparse matrix from the defined equation. The following code snippet illustrates the process for a simple Poisson equation:

```
Assembler<double> assembler(problem);
Variable var_phi;

assembler.defineEquation(partvol * rho +
   eps0 * area(epsr * grad(var_phi)));

Sparse<double> matrix(problem.size());
Full<double> rhs(problem.size(), 1);
assembler.assembleLinear(matrix, rhs);
```

The symbolic `Sparse` matrix is converted to a CSR/CSC format [49] (`ConstSparse`) which can be processed using matrix operations and solvers displayed in Section 4.3.

6.3 Numerical Solvers

Two classes of numerical solvers are crucial for quantum-electronic simulation: linear solvers and eigenvalue solvers. Both are provided by a number of software packages that are stable and efficient. VSP links to several numerical libraries and tries to select the most appropriate solver for a problem at hand during run-time.

Available linear solvers in VSP are divided into (i) dense solvers provided by LAPACK or an interface-compatible library, (ii) direct sparse solvers such as SuperLU [50] or PARDISO [51], and (iii) iterative solvers such as CG, GMRES, or BiCGStab [49]. The linear solver selection follows the rules:

- Full LAPACK for very small systems
- Banded LAPACK for 1D problems
- Direct sparse for 2D problems
- ILU-preconditioned iterative for 3D problems

Available eigenvalue solvers fall into two categories: "direct" solvers provided by LAPACK and subspace solvers such as IRAM provided by ARPACK or Jacobi-Davidson [52]. The *efficiency* of the eigenvalue solver is crucial, since most of the simulation time in quantum problems is spent there. The eigenvalue solver is selected based on the following rules:

- Full LAPACK for very small systems (e.g. bulk $\mathbf{k}\cdot\mathbf{p}$)
- Banded LAPACK for 1D systems with few variables
- ARPACK (shift-invert) for 1D/2D problems
- ARPACK (plain) for definite 3D problems

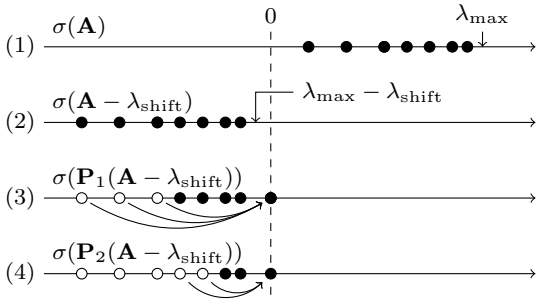


Fig. 18 Searching for eigenvalues up to to $\lambda_{\max}$; (1) shows the spectrum of a positive definite Hermitian matrix $\mathbf{A}$. $\mathbf{A}$ is first shifted to the left by $\lambda_{\text{shift}} > \lambda_{\max}$ (2) and the first $n_{\text{ev}} = 3$ eigenvalues computed by a subspace solver (e.g. ARPACK); a projection matrix $\mathbf{P}_1 = \mathbf{I} - \mathbf{v}_i\mathbf{v}_i^{\text{H}}$ is constructed from the eigenvectors $\mathbf{v}_i$. The subspace solver is invoked again on the projected system $\mathbf{P}_1(\mathbf{A} - E_{\text{shift}})$. The projection (3) moves the found eigenvalues to 0 and effectively prevents the solver to converge on already found eigenvalues. The process is repeated (4) until all eigenvalues $< \lambda_{\max}$ are found.

- Jacobi-Davidson for large/indefinite 3D problems

One challenge in quantum-electronic carrier models described in Section 5.3.2 and Section 5.3.4 is that the exact number of eigenvalues that is needed is unknown beforehand; instead, the carrier model requires all eigenvalues within an energy interval up to E_{lim} as indicated in Fig. 13. While this can be done using direct solvers with hardly any performance penalty, subspace solvers require the number of sought eigenvalues to be known. We resolve this issue using a technique called *subspace deflation* explained in Fig. 18. The method extends the operation of existing subspace-based eigenvalue solvers without the need to alter their code.

6.4 Numerical Integration

We researched various (adaptive) integration methods for the use of energy integration in the NEGF formalism. These methods are outlined in the following sections. The Clenshaw-Curtis integration also proved useful for $\mathbf{k}$-space integration of electronic subbands [34].

6.4.1 Polynomial Interpolation

Simpson's rule is based on equidistant grid points and an interpolation polynomial of second order. As an alternative, a more general approach with non-equidistant grid points and polynomials of arbitrary degree can be considered. For a monomial power basis the interpolation polynomial on N nodes takes the form

$$p(x) = \sum_{i=1}^{N} a_i x^{i-1}. \tag{19}$$

To obtain the coefficient vector $\mathbf{a} = [a_1, a_2, a_3, \ldots, a_N]^T$ an equation system of rank N needs to be solved

$$\underbrace{\begin{bmatrix} 1 & x_1 & \cdots & x_1^{N-1} \\ 1 & x_2 & \cdots & x_2^{N-1} \\ \vdots & & \ddots & \vdots \\ 1 & x_N & \cdots & x_N^{N-1} \end{bmatrix}}_{\mathbf{V}} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_N \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}, \tag{20}$$

where $\mathbf{V}$ is called the Vandermonde matrix. Unfortunately this system is often ill-conditioned and its solution may become numerically unstable. Björck and Pereyra [53] developed an algorithm that is able to calculate the coefficient vector in a fast and stable manner.

After the coefficients of the polynomial are obtained the integral of the interpolation function in the interval $[x_1, x_N]$ can be calculated. For an arbitrary odd number N of grid points, a subset of $(N+1)/2$ grid points may be used to obtain a second polynomial and consequentially a second approximation of the integral. These two results are then compared to yield the error criterion for the adaptive integration algorithm. Unfortunately, polynomial interpolation functions on equidistant points suffer from Runge's phenomenon for a higher degree. This can be avoided using non-equidistant grid points as done by the Clenshaw-Curtis rule.

6.4.2 Clenshaw-Curtis Integration

Fejér [54] proposed to use the zeros of the Chebyshev polynomial $T_n = \cos(n \arccos x)$ in the interval $]-1, 1[$ as quadrature points of the integral of $f(x)$,

$$\int_{-1}^{1} f(x)\mathrm{d}x = \sum_{k=0}^{n} w_k f(x_k). \tag{21}$$

For Fejér's second rule, the $n-1$ extreme points of T_n are used. Clenshaw and Curtis [55] extended this open rule to a closed form which includes the boundary points $x_0 = -1$ and $x_n = 1$ of the interval. The $n+1$ quadrature points are $x_k := \cos(\vartheta_k)$ with $\vartheta_k := k\frac{\pi}{n}$ and $k = 0, 1, \ldots, n$. The weights w_k in equation (21) are to be obtained by an explicit expression or by means of discrete Fourier transforms [56]. The explicit expressions of the Clenshaw-Curtis weights are:

$$w_k = \frac{c_k}{n}\left(1 - \sum_{j=1}^{\lfloor n/2 \rfloor} \frac{b_j}{4j^2 - 1}\cos(2j\vartheta_k)\right). \tag{22}$$

The coefficients b_j and c_k are given by

$$b_j = \begin{cases} 1, \text{ if } j = n/2 \\ 2, \text{ if } j < n/2 \end{cases}, \quad c_k = \begin{cases} 1, \text{ if } k = 0 \mod n \\ 2, \text{ otherwise.} \end{cases}$$

A useful property of the Clenshaw-Curtis rule is the option to create subsets of the quadrature nodes. To move from $n+1$ to $2n+1$ points only n new function values need to be evaluated.

6.4.3 Extended Doubly Adaptive Quadrature Routine

So far the presented methods used a fixed rule with interval refinement for adaptive energy integration. A different approach, which comprises two refinement strategies, has been presented by Espelid [57]. A global error criterion is used to find the most erroneous subinterval. This interval is then treated locally either by subdivision and applying a smaller order Newton-Cotes rule, or by inserting additional energy grid points and using a higher order rule, depending on the estimated error. The local integral and error of the superior method for a given subinterval are then added to the global values. This procedure is repeated until the global error is below a given tolerance.

6.5 Performance and Parallelization

The simulator's infrastructure and its models are designed to run on workstation computers. VSP makes use of parallelization on shared-memory systems. Runtime naturally depends on device size and mesh density, number of variables, and the number of usable cores. It has been taken care to achieve optimal scaling although this is strongly problem-dependent: most models have a linear and a (typically small) quadratic run-time contribution with respect to overall problem size.

7 Customization – Software Development Kit

Next to the customization of the work flow through the **Chain** model and user extensions of the *material database* we provide a means of customization on the model level through the VSP software development kit (SDK). The user implemented model needs to fulfill a minimal interface to be compatible with VSP

```
class ModuleTemplate : public ModelExtended {
  // quantities
  Quan<double>           quanDouble;
  Quan<double, Segment> quanDoubleOnSegment;
  ...

public:
  ModuleTemplate(ObjectUnstructured *obj);

  void solve(); // called at a model run

  const static string classId; // model name
};
```

The **ModuleTemplate** inherits the necessary functionality from the **ModelExtended**. The user can add *quantities*, *properties*, and *parameters* and manipulate them at input through the device file and the IPD. When the model is run either as submodel, through the **Chain** or the IPD, the *solve* method gets called. There the functionality of the model is implemented. All the standard C++ language features including use of third-party libraries are available to the developer. Furthermore, the user has full access to the VSP numerical libraries and the modeling language comprised of functionals for compile-time calculations. It is possible to obtain *quantities* from other models, add and run submodels, access and use the linear and eigenvalue solvers, access the device information and so on. Models implemented with the SDK get compiled into a library file, that is loaded by the VSP at run time.

An exemplary application would be a user-defined carrier model that takes the potential as input and has the carrier densities as output. Such a model is compatible with the **SCLoop** model and can be used the same way as carrier models already provided with VSP.

8 Simulation Examples

As shown in the previous sections VSP is controlled through the input deck. The typical work flow consists of device generation, setting the simulation parameters, running VSP, and visualization. A convenient graphical user interface for the complete work flow including the manipulation of the IPD is provided by the *GTS Framework* [58]. In the following we present some simulation examples of the VSP.

8.1 Conductivity in Fin FETs

The GTS framework [58] was used to create a device structure and mesh from a transmission electron microscope (TEM) image of an nMOS fin FET by Intel [59] (Fig. 19). We calculated the low-field conductivity of the channel from the linearized Boltzmann transport equation (L-BTE) solved using the Kubo-Greenwood formalism. Isotropic phonon and surface roughness scattering (SRS) are included at 300 K. A novel SRS model [60, 61] is employed; it extends the model of Prange and Nee [62, 63] with respect to non-planar geometries and band anisotropy and thus gives results consistent with planar geometries.

The transfer characteristic of the device for different combinations of channel and substrate orientation is shown in Fig. 20. The channel conductivity severely degrades for the orientations [110]/(001) and [110]/(1$\bar{1}$1) but not for [110]/(1$\bar{1}$0) and [100]/(010). A Coulomb scattering model is also available; it handles both screened ionized impurities in the channel and unscreened charges in the oxide. However, Coulomb scattering has not been included in the study shown in Fig. 20.

Although the perturbative Kubo-Greenwood method has its shortcomings (lack of carrier heating, short channel effects) the model has low computational demands while taking care of a good number of factors that influence transport (confinement, strain, orientation, temperature, etc.). This allows quick evaluation of channel designs for engineering purposes while maintaining runtime below a minute per bias point on a workstation computer even for problems of considerable size.

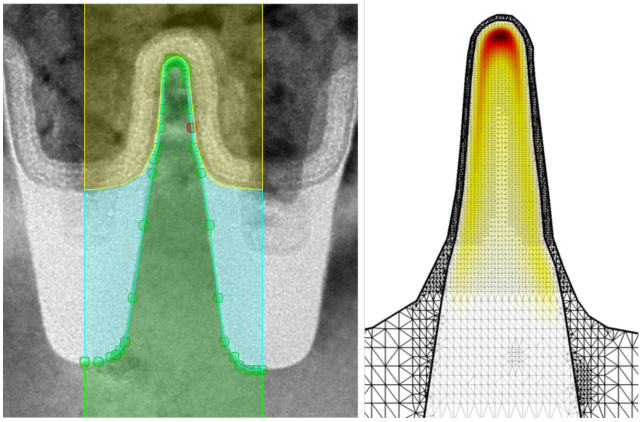


Fig. 19 Left: TEM image of nMOS fin structure fabricated by Intel [59]; segments of simulation domain are overlaid. Right: electron concentration in fin under gate bias $V_G = 1$ V from self-consistent Schrödinger-Poisson calculation; the computational grid is shown.

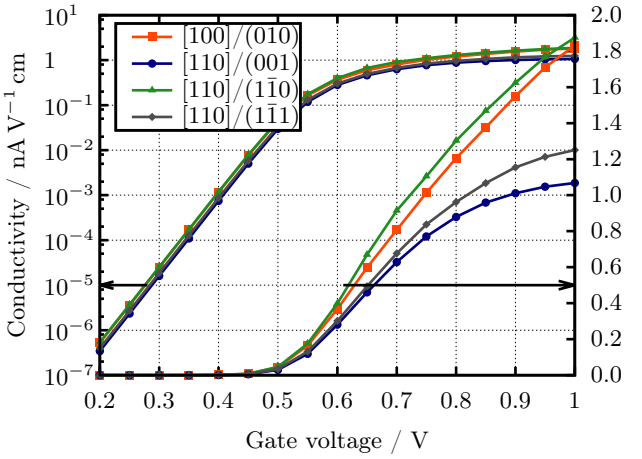


Fig. 20 Fin channel transfer characteristic for four different channel/substrate orientations of the device in Fig. 19; degradation of the characteristic can be observed for [110]/(001) and [110]/(1$\bar{1}$1) orientations, but not for [110]/(1$\bar{1}$0) which has about the same drive current as [100](010), the traditional orientation in Si MOSFET fabrication.

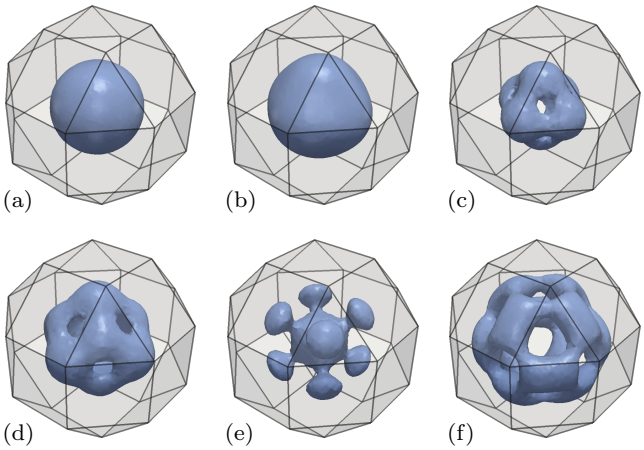


Fig. 21 Contours of densities corresponding to quantized hole states in a silicon quantum dot calculated with the six-band **k**·**p** model; densities of degenerate states are summed in this figure; (a) states 0 through 5, (b) states 6 through 11, (c) states 12 through 15, (d) states 16 through 21, (e) states 22 through 27, (f) states 28 and 29

8.2 Confined States in Quantum Dots

To demonstrate the 3D capabilities of VSP we simulated a silicon quantum dot embedded in a SiO_2/Si_3N_4 matrix, a structure commonly found in photoluminescence experiments [64]. The dot has the geometric shape of a rhombicuboctahedron and measures 5 nm along its ⟨100⟩ axes. Closed (Dirichlet) boundary conditions were applied around the dot.

Most of the calculated states are degenerate due to the symmetric structure geometry. The most common degeneracy multiplicity is six which corresponds to the number of bands (three) times spin polarizations (two). Multiplicities other than six are due to non-parabolicity effects. Spin-orbit coupling partially lifts the six-fold degeneracies and breaks them into two- and fourfold.

Figure 21 shows the densities corresponding to each cluster of degenerate states. The states develop very unusual forms due to the non-parabolicity of the Hamiltonian. The densities are almost unaffected by spin-orbit coupling; the densities with and without spin-orbit coupling are visually indistinguishable.

8.3 Non-Equilibrium Green's Functions

A well known example for a multi-physics and multi-scale simulation is the RTD model of NEMO-1D [24,65]. It is comprised of extended contact regions with semi-classical carriers, an equilibrium quantum region around the barriers and a small non-equilibrium region where transport is calculated through the barriers and the well. Such a model can be easily set up in VSP.

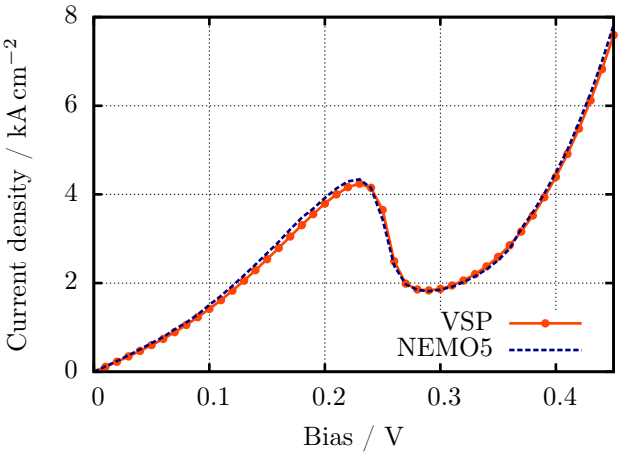


Fig. 22 I/V characteristics of a simple RTD; Comparison of VSP result to default example of NEMO5 on nanohub [65,66]

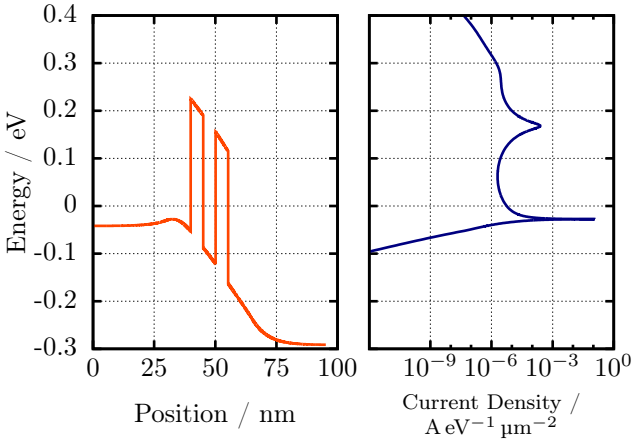


Fig. 23 Conduction band edge and energetically resolved current density of the RTD in Fig. 22 biased at 0.25 V

We calculated the I/V of a $GaAs/Al_{0.3}Ga_{0.7}As$ RTD with 5 nm barrier and well width (Fig. 22). This is the default example template provided by [66] and the results of NEMO5 are given for comparison. The conduction band edge and energetically resolved current density in Fig. 23 show the first current-carrying quasi-bound state. The whole I/V calculation takes less than 30 s on a single CPU core.

A very interesting application of VSPs NEGF model is the investigation of metal-oxide-semiconductor (MOS) devices. To model the broadening of the quasi-bound states we use an optical potential [65]. Multiple runs of the NEGF base model are carried out to consider all the material's valleys and sum up the contributions. The model gives insight to macroscopic and microscopic properties of high-k metal gate stacks in a consistent manner [38]. The local density of occupied states of an exemplary nMOS device with $t_{ox} = 1.2\,\text{nm}$ and bulk doping $N_A = 3 \times 10^{17}\,\text{cm}^{-3}$ at a gate voltage of 1 V is

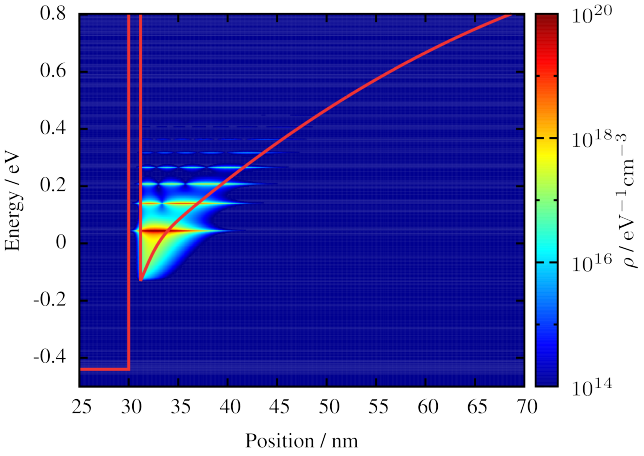


Fig. 24 Self-consistent calculation of the local density of occupied states in an nMOS device with $t_{ox} = 1.2\,\text{nm}$ at a gate voltage of 1 V with the 1D NEGF model

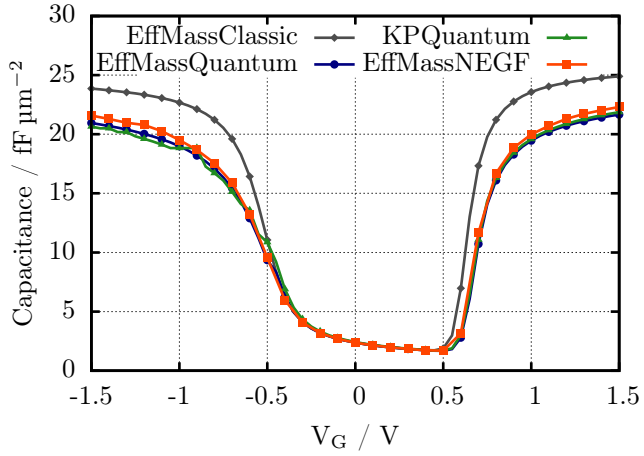


Fig. 25 Capacitance-voltage characteristics of an nMOS device calculated with different carrier models available in VSP

shown in Fig. 24. The calculated C/V (Fig. 25) of this device for all the carrier models presented in Section 5.3 show excellent agreement for all quantum-mechanical models considering their different physical aspects.

The ballistic 2D NEGF model has multi-terminal support and relies on the recursive Green's function algorithm by Svizhenko [28]. It is well suited for investigation of nano MOS devices. An nMOS with 25 nm effective gate length and $t_{ox} = 1.5\,\text{nm}$ was used as example device. The self-consistently calculated conduction band edge and electron concentration at $V_G = 1\,\text{V}$ and $V_{SD} = 0.3\,\text{V}$ are shown in Fig. 26. Calculated quantities were visualized with the graphical user interface *GTS Vision* [58]. The I/V characteristics of the device are depicted in Fig. 27. Two-terminal devices such as a quantized resonant tunneling diode can be modeled the same way. Figure 28 shows the local density of states

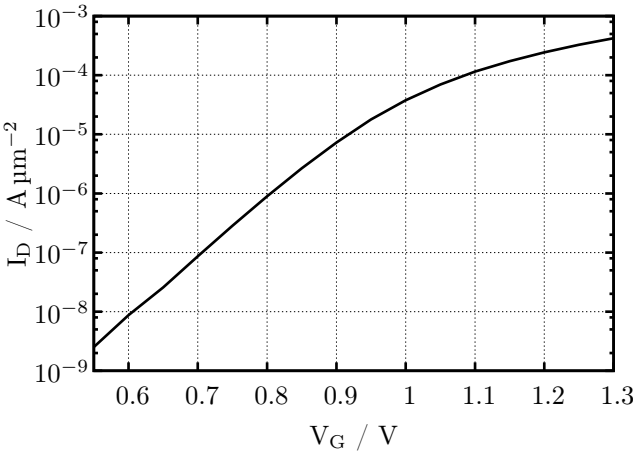


Fig. 27 I/V characteristics of the nMOS in Fig. 26

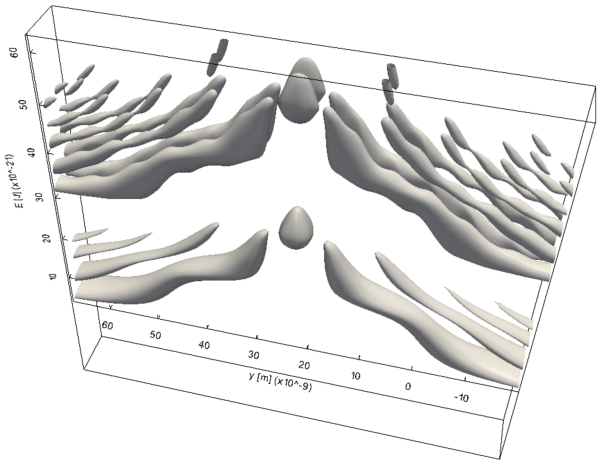


Fig. 28 Self-consistently calculated density of states of a quantized resonant tunneling diode calculated with 2D NEGF

of such a device with a clearly visible resonant state in the well.

The 1D/2D NEGF calculations are shared-memory parallelized with respect to the energy grid. This results in near linear scaling and is especially useful in 2D.

8.4 Quantum Cascade Detector

One of the essential technologies in modern photonic systems are semiconductor heterostructures. The first use of a QCL as a photo-detector has been reported by [67] and was since then refined for infrared and terahertz wavelengths [68] leading to the current quantum cascade detector (QCD). The operating principle of a QCD is as following. A ground level electron is excited to a higher state by absorbing a photon. Due to the asymmetric design, the electron relaxes in a preferred direction into the quantum well of the next cascade. This concept reduces dark current and noise.

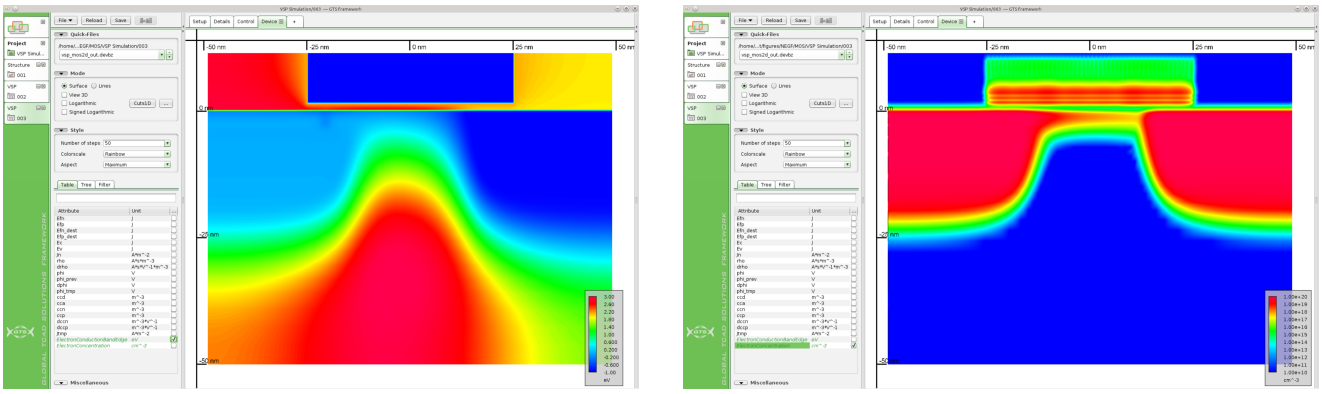


Fig. 26 Self-consistent conduction band edge (left) and electron concentration (right) of a 2D nMOS with 25 nm gate length calculated with NEGF at $V_G = 1\,V$ and $V_{SD} = 0.3\,V$; Visualization is done with the graphical user interface from GTS [58]

We use the semi-classical Pauli master equation as outlined in Section 5.4 to model electron current transport through the multi-layer semiconductor heterostructure. The incorporated model for stimulated emission and absorption of photons is essential for the description of a QCD. As an example device we use a mid-infrared QCD operating at a wavelength of 4.7 µm. The design of the InGaAs/InAlAs QCD is taken from [68] (device N1022) and all simulation results are compared to the measurements therein. We calculate the responsivity, which relates the incoming photon flux to the detected current with the four-band **k·p** model including in-plane nonparabolicity [69]. It shows excellent agreement with measurement (cf. Fig. 29).

For QCL and QCD devices we use a Poisson solver with periodic boundary conditions, thereby only considering a single device period for the calculation of the potential. We provide self-consistency of the potential with an equilibrium occupation of the subbands as an initial guess for the MC solver. Full self-consistency of the subband populations from the PME, the Schrödinger solver and the electrostatic potential is available too. As expected for QCDs the fully self-consistent calculation only has a small impact on the results (Fig. 30).

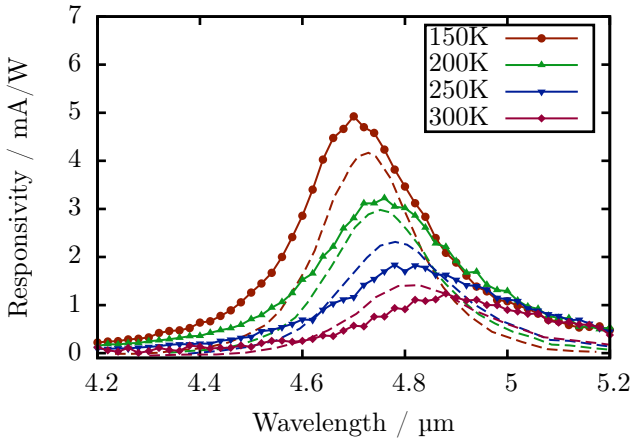


Fig. 29 Calculated responsivity (solid) of the QCD compared to measurements of [68] (dashed); Cross-plane band structure modeled with a four-band **k·p** Hamiltonian; in-plane dispersion in the transport model is assumed non-parabolic with the effective mass and nonparabolicity coefficient fitted to the subband structure; inclusion of nonparabolicity has substantial effects on the responsivity.

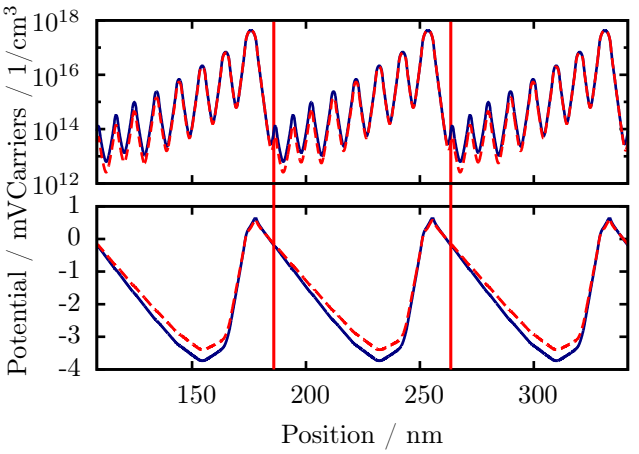


Fig. 30 Self-consistent populations vs. non-self-consistent population (dashed) of a QCD; the upper plot shows the calculated carrier density; the lower plot the according potential;

9 Availability

VSP is developed collaboratively by the Institute for Microelectronics, TU Wien and Global TCAD Solutions LLC (GTS). VSP can be freely tested and evaluated online through the MyGTS web portal [70]. Commercial and academic licenses of VSP are distributed by GTS [71]. The executable is available for Linux and Windows platforms. An extensive graphical user interface for 2D/3D device generation, simulation setup, and visualization is provided through the GTS framework.

10 Conclusion

In this paper we present VSP (Vienna Schrödinger-Poisson), a simulation framework for quantum-electronic and nano-physical applications. We developed a flexible tool with a high level of automation. Individual models can be easily combined to create a specific work flow. Data can be transfered and shared between the models. VSP provides access to efficient numerical methods and libraries. The development effort was guided by the objective of a consistent treatment of nanoelectronic TCAD problems. An abstract modeling language enables the simultaneous definition of problems in 1D, 2D, and 3D. User customization through a software development kit allows a simple integration of new models with full access to the VSP core functionality, abstract modeling, solvers, and input/output functions.

We demonstrate the self-consistent calculation of carriers and states in MOS and RTD devices using different physical models. Transport calculations were carried out for these devices using NEGF. Semi-classical transport modeling with the Pauli master equation is shown for quantum cascade detectors.

The outstanding features of VSP include the user-defined model structure with fully configurable control and data flow and the problem separation in data, topology and modeling levels. It is accessible through a software development kit shipped with VSP. We provide in-source documentation based on *literate modeling*. The tool makes use of unstructured meshes and operates with arbitrary geometries to provide consistent models in all dimensions. VSP is able to treat arbitrary crystal orientations and provide automated detection of confined and free carriers in closed-boundary problems.

VSP is aimed at researchers, engineers and students in the field of nanoelectronics. With VSP, we wish to provide a simulation framework that is extensible and adjustable to a wide range of engineering problems.

Acknowledgements This work has been supported by the Austrian Science Fund program F025 (IR-ON), and the Austrian Research Promotion Agency, project 838551 (NeGFQTS).

References

1. Kannan, G., Vasileska, D.: In: Computational Electronics (IWCE), 2010 14th International Workshop on, pp. 1 –4 (2010). doi:10.1109/IWCE.2010.5677977
2. Tan, I.H., Snider, G.L., Chang, L.D., Hu, E.L.: J. Appl. Phys. **68**(8), 4071–4076 (1990). http://link.aip.org/link/?JAP/68/4071/1
3. Snider, G.L.: (2013). http://www.nd.edu/~gsnider/
4. Birner, S., Zibold, T., Andlauer, T., Kubis, T., Sabathil, M., Trellakis, A., Vogl, P.: IEEE T. Electron. Dev. **54**(9), 2137 –2142 (2007). doi:10.1109/TED.2007.902871
5. Trellakis, A., Zibold, T., Andlauer, T., Birner, S., Smith, R., Morschl, R., Vogl, P.: Journal of Computational Electronics **5**, 285–289 (2006). http://dx.doi.org/10.1007/s10825-006-0005-x. 10.1007/s10825-006-0005-x
6. Steiger, S.: NEMO 5 User Manual. NCN Purdue Univ.
7. Steiger, S., Povolotskyi, M., Hong-Hyun Park, Kubis, T., Klimeck, G.: IEEE Transactions on Nanotechnology, **10**(6), 1464–1474 (2011). doi:10.1109/TNANO.2011.2166164
8. Auf der Maur, M., Penazzi, G., Romano, G., Sacconi, F., Pecchia, A., Di Carlo, A.: IEEE T. Electron. Dev. **58**(5), 1425 –1432 (2011). doi:10.1109/TED.2011.2114666
9. Auf der Maur, M., Sacconi, F., Penazzi, G., Romano, G., Povolotskyi, M., Pecchia, A., Di Carlo, A.: Journal of Computational Electronics **9**, 262–268 (2010). http://dx.doi.org/10.1007/s10825-010-0331-x. 10.1007/s10825-010-0331-x
10. Steiger, S., Veprek, R., Witzigmann, B.: Optical and Quantum Electronics **41**, 551–557 (2009). http://dx.doi.org/10.1007/s11082-010-9360-8. 10.1007/s11082-010-9360-8
11. Veprek, R.G.: Computational Modeling of Semiconductor Nanostructures for Optoelectronics. Ph.D. thesis, ETH Zürich (2009)
12. SILVACO, Inc.: ATLAS User's Manual
13. Sentaurus Device. http://www.synopsys.com
14. Karner, M., Gehring, A., Holzer, S., Pourfath, M., Wagner, M., Goes, W., Vasicek, M., Baumgartner, O., Kernstock, C., Schnass, K., Zeiler, G., Grasser, T., Kosina, H., Selberherr, S.: Journal of Computational Electronics **6**(1), 179–182 (2007). doi:10.1007/s10825-006-0077-7
15. Stanojevic, Z., Kosina, H.: In: IWCE, pp. 93–94 (2013)
16. Klima, R., Grasser, T., Selberherr, S., Wien, T.: In: 15th European Simulation Multiconference, pp. 161–165 (2001)
17. Avila, L.S., Barre, S., Blue, R., Geveci, B., Henderson, A., Hoffman, W.A., King, B., Law, C.C., Martin, K.M., Schroeder, W.J.: The VTK User's Guide. Kitware (2010)
18. Knuth, D.E.: The Computer Journal **27**(2), 97–111 (1984)
19. OASIS Darwin Information Typing Architecture (DITA) Version 1.2 Specification (2010). http://docs.oasis-open.org/dita/v1.2/spec/DITA1.2-spec.html
20. Fischer, C.: Bauelementsimulation in einer computergestützten Entwurfsumgebung. Ph.D. thesis, Technische Universität Wien (1994)
21. Institute for Microelectronics and Global TCAD Solutions GmbH: Minimos-NT User Manual. http://www.globaltcad.com/en/products/minimos-nt.html
22. Markus Karner: Multi-Dimensional Simulation of Closed Quantum Systems. Master's thesis, Technische Universität Wien (2004)
23. Trellakis, A., Galick, A.T., Pacelli, A., Ravaioli, U.: J. Appl. Phys. **81**(12), 7880–7884 (1997). doi:10.1063/1.365396. http://link.aip.org/link/?JAP/81/7880/1
24. Lake, R., Klimeck, G., Bowen, R.C., Jovanovic, D.: J. Appl. Phys. **81**(12), 7845–7869 (1997)
25. Chris Bowen, R., Gerhard Klimeck, Roger K. Lake, William R. Frensley, Ted Moise: J. Appl. Phys. **81**(7), 3207–3213 (1997)
26. Cresti, A., Farchioni, R., Grosso, G., Pastori Parravicini, G.: Phys. Rev. B **68**, 075,306 (2003)
27. Pourfath, M., Kosina, H., Selberherr, S.: J. Comput. Electron. **5**, 155–159 (2006)
28. Svizhenko, A., Anantram, M.P., Govindan, T.R., Biegel, B., Venugopal, R.: J. Appl. Phys. **91**(4), 2343–2354 (2002)
29. John, D., Castro, L., Pereira, P., Pulfrey, D.: In: Nanotech, vol. 3, pp. 65–68 (2004)

30. Baumgartner, O., Karner, M., Holzer, S., Pourfath, M., Grasser, T., Kosina, H.: In: Proceedings of the 2007 NSTI Nanotechnology Conference, vol. 3, pp. 145–148 (2007)
31. Ikonić, Z.: In: P. Harrison (ed.) Quantum Wells, Wires and Dots, pp. 345–369. Wiley Interscience (2005)
32. Baumgartner, O., Karner, M., Sverdlov, V., Kosina, H.: In: 13th International Workshop on Computational Electronics (IWCE), pp. 53–56 (2009). doi:10.1109/IWCE.2009.5091131
33. Luttinger, J.M., Kohn, W.: Physical Review **97**(4), 869 (1955). http://link.aps.org/abstract/PR/v97/p869
34. Baumgartner, O., Karner, M., Sverdlov, V., Kosina, H.: Solid-State Electronics **54**(2), 143–148 (2010). doi:10.1016/j.sse.2009.12.010. http://www.sciencedirect.com/science/article/pii/S0038110109003530. Selected Full-Length Extended Papers from the EUROSOI 2009 Conference
35. Stanojevic, Z., Baumgartner, O., Sverdlov, V., Kosina, H.: In: Proceedings of the 14th International Workshop on Computational Electronics (IWCE), pp. 5–8 (2010). doi:10.1109/IWCE.2010.5677927
36. Stanojevic, Z., Sverdlov, V., Baumgartner, O., Kosina, H.: Solid State Electron. **70**(0), 73 – 80 (2012). doi:10.1016/j.sse.2011.11.022. http://www.sciencedirect.com/science/article/pii/S0038110111004187
37. Hensel, J.C., Hasegawa, H., Nakayama, M.: Phys. Rev. **138**(1A), A225–A238 (1965). http://link.aps.org/abstract/PR/v138/pA225
38. Baumgartner, O., Stanojevic, Z., Kosina, H.: In: Proceedings of the 16th International Conference on Simulation of Semiconductor Processes and Devices (SISPAD), pp. 91–94 (2011)
39. Sirtori, C., Capasso, F., Faist, J., Scandolo, S.: Phys. Rev. B **50**(12), 8663–8674 (1994). http://link.aps.org/abstract/PRB/v50/p8663
40. Bahder, T.B.: Phys. Rev. B **41**(17), 11,992–12,001 (1990). http://link.aps.org/abstract/PRB/v41/p11992
41. Kriechbaum, M., Ambrosch, K.E., Fantner, E.J., Clemens, H., Bauer, G.: Phys. Rev. B **30**, 3394–3405 (1984). doi:10.1103/PhysRevB.30.3394. http://link.aps.org/doi/10.1103/PhysRevB.30.3394
42. Iotti, R.C., Rossi, F.: Phys. Rev. Lett. **87**(14), 146,603 (2001). http://link.aps.org/abstract/PRL/v87/e146603
43. Jirauschek, C., Scarpa, G., Lugli, P., Vitiello, M.S., Scamarcio, G.: J. Appl. Phys. **101**(8), 086109 (2007). doi:10.1063/1.2719683. http://link.aip.org/link/?JAP/101/086109/1
44. Milovanovic, G., Kosina, H.: J. Comput. Electron. **9**, 211–217 (2010). doi:10.1007/s10825-010-0325-8
45. Iotti, R.C., Ciancio, E., Rossi, F.: Phys. Rev. B **72**(12), 125,347 (2005). doi:10.1103/PhysRevB.72.125347
46. Baumgartner, O., Stanojevic, Z., Kosina, H.: In: K.K. Sabelfeld, I. Dimov (eds.) Monte Carlo Methods and Applications, De Gruyter Proceedings in Mathematics, chap. 7, pp. 59–67. De Gruyter (2012)
47. Stanojevic, Z., Karner, M., Schnass, K., Kernstock, C., Baumgartner, O., Kosina, H.: In: Proceedings of the 16th International Conference on Simulation of Semiconductor Processes and Devices (SISPAD), pp. 143–146 (2011)
48. Kramer, K.M., Hitchon, W.N.G.: Semiconductor Devices. Prentice Hall (1997)
49. Saad, Y.: Iterative Methods for Sparse Linear Systems. PWS Publishing Company (1996)
50. Li, X.S.: ACM Transactions on Mathematical Software **31**(3), 302–325 (2005)
51. Schenk, O., Bollhöfer, M., Römer, R.: SIAM Review **50**(1), 91–112 (2008). doi:10.1137/070707002. http://epubs.siam.org/doi/abs/10.1137/070707002
52. Bai, Z., Demmel, J., Dongarra, J., Ruhe, A., van der Vorst, H. (eds.): Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (2000)
53. Björck, A., Pereyra, V.: Math. Comp. **24**(112), 893–903 (1970). doi:10.2307/2004623
54. Fejér, L.: Math. Z. **37**, 287–309 (1933)
55. Clenshaw, C.W., Curtis, A.R.: Num. Math. **2**, 197—205 (1960)
56. Waldvogel, J.: BIT **46**, 195–202 (2006). doi:10.1007/s10543-006-0045-4
57. Espelid, T.: BIT **43**(2), 319–337 (2003). doi:10.1023/A:1026087703168
58. GTS Framework. http://www.globaltcad.com/en/products/gts-framework.html
59. Auth, C., Allen, C., Blattner, A., Bergstrom, D., Brazier, M., Bost, M., Buehler, M., Chikarmane, V., Ghani, T., Glassman, T., Grover, R., Han, W., Hanken, D., Hattendorf, M., Hentges, P., Heussner, R., Hicks, J., Ingerly, D., Jain, P., Jaloviar, S., James, R., Jones, D., Jopling, J., Joshi, S., Kenyon, C., Liu, H., McFadden, R., Mcintyre, B., Neirynck, J., Parker, C., Pipes, L., Post, I., Pradhan, S., Prince, M., Ramey, S., Reynolds, T., Roesler, J., Sandford, J., Seiple, J., Smith, P., Thomas, C., Towner, D., Troeger, T., Weber, C., Yashar, P., Zawadzki, K., Mistry, K.: In: VLSIT, pp. 131–132 (2012). doi:10.1109/VLSIT.2012.6242496
60. Stanojevic, Z., Kosina, H.: In: Silicon Nanoelectronics Workshop, pp. 132–133 (2013)
61. Stanojevic, Z., Kosina, H.: In: Intl. Conf. on Simulation of Semiconductor Processes and Devices (SISPAD) (2013)
62. Prange, R.E., Nee, T.W.: Phys. Rev. **168**, 779–786 (1968). doi:10.1103/PhysRev.168.779. http://link.aps.org/doi/10.1103/PhysRev.168.779
63. Jin, S., Fischetti, M.V., Tang, T.w.: J. Appl. Phys. **102**(8), 083715 (2007). doi:10.1063/1.2802586. http://link.aip.org/link/?JAP/102/083715/1
64. Kang, Z.T., Arnold, B., Summers, C.J., Wagner, B.K.: Nanotechnology **17**(17), 4477 (2006). http://stacks.iop.org/0957-4484/17/i=17/a=032
65. Klimeck, G., Lake, R., Bowen, R.C., Frensley, W.R., Moise, T.S.: Appl. Phys. Lett. **67**(17), 2539–2541 (1995)
66. Park, H.H., Jiang, Z., Akkala, A.G., Steiger, S., Povolotskyi, M., Kubis, T.C., Sellier, J.M.D., Tan, Y., Kim, S., Luisier, M., Agarwal, S., McLennan, M., Klimeck, G., Geng, J.: Resonant Tunneling Diode Simulation with NEGF (2008). doi:10.4231/D3DZ03144. https://nanohub.org/resources/5237
67. Hofstetter, D., Beck, M., Faist, J.: Appl. Phys. Lett. **81**(15), 2683–2685 (2002). doi:10.1063/1.1512954. http://link.aip.org/link/?APL/81/2683/1
68. Giorgetta, F., Baumann, E., Graf, M., Quankui Yang, Manz, C., Kohler, K., Beere, H., Ritchie, D., Linfield, E., Davies, A., Fedoryshyn, Y., Jackel, H., Fischer, M., Faist, J., Hofstetter, D.: IEEE Journal of Quantum Electronics **45**(8), 1039–1052 (2009). doi:10.1109/JQE.2009.2017929
69. Baumgartner, O., Stanojevic, Z., Kosina, H.: In: 16th International Workshop on Computational Electronics (IWCE), pp. 86–87 (2013)
70. MyGTS. www.globaltcad.com/mygts
71. Vienna Schrödinger Poisson. http://www.globaltcad.com/vsp