GQLink

An implementation of Quantized State System (QSS) methods in Geant4

Lucio Santi[1], Rodrigo Castro[1,2]
Soon Yung Jun, Krzysztof Genser, Daniel Elvira[3]
[1]Universidad de Buenos Aires [2]ICC-CONICET [3]FNAL

Geant4 21st Collaboration Meeting
12-16 September 2016

Table of contents

Introduction

Motivation of this work

- Simulation in HEP involves numerical solutions to ODE systems in order to determine the trajectories described by charged particles in a magnetic field.
- As a particle moves through a detector, each volume crossing interrupts the underlying numerical solver.
- Traditional methods invest considerable computational efforts to handle these discontinuities accurately (detection of intersection points).

Motivation of this work

- **Quantized State System** methods (**QSS**, Kofman 2001 [4]) are a novel family of numerical integration methods exhibiting attractive features for this type of HEP simulation.
- The goals pursued in this work are:
  - To develop a proof-of-concept implementation of QSS within Geant4,
  - To address its suitability as an alternative production integrator, and
  - To evaluate its performance in a realistic HEP application.

Quantized State System (QSS) methods

Quantized State System methods

- QSS methods are based on **state variable quantization**.
- As opposed to traditional solvers which discretize time (e.g., Runge-Kutta family) QSS discretizes the system's state.
- **State variables** are thus approximated by **quantized variables**.
- The relation between both is given by a **quantization function** which is in charge of the **accuracy control**.

ODE system		**Quantized system**
$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t))$	$\Rightarrow$	$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{q}(t))$

QSS: motivational example (perfect oscillator)

Original system

$$\begin{cases} \dot{x}_1(t) = x_2(t) \\ \dot{x}_2(t) = -x_1(t) \end{cases}$$

Initial conditions

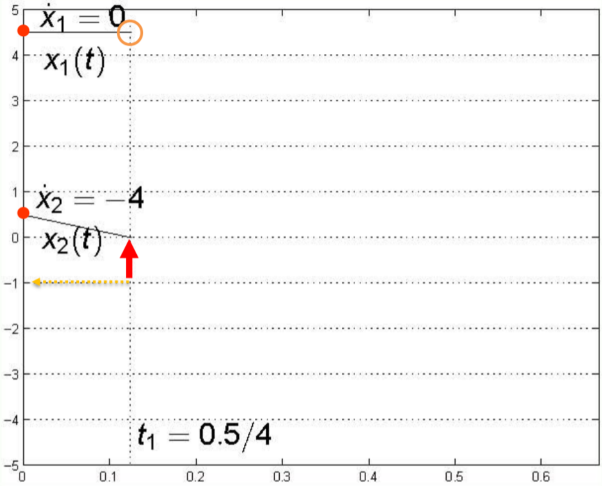
$$\begin{cases} x_1(0) = 4.5 \\ x_2(0) = 0.5 \end{cases}$$

Quantization function

$$q_i(t) = \lfloor x_i(t) \rfloor$$

Quantized system

$$\begin{cases} \dot{x}_1(t) = q_2(t) \\ \dot{x}_2(t) = -q_1(t) \end{cases}$$



QSS: motivational example (perfect oscillator)

Original system

$$\begin{cases} \dot{x}_1(t) = x_2(t) \\ \dot{x}_2(t) = -x_1(t) \end{cases}$$

Initial conditions

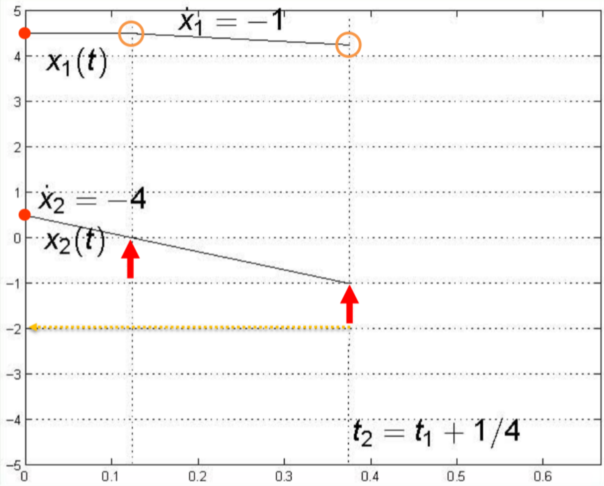
$$\begin{cases} x_1(0) = 4.5 \\ x_2(0) = 0.5 \end{cases}$$

Quantization function

$$q_i(t) = \lfloor x_i(t) \rfloor$$

Quantized system

$$\begin{cases} \dot{x}_1(t) = q_2(t) \\ \dot{x}_2(t) = -q_1(t) \end{cases}$$



QSS: motivational example (perfect oscillator)

Original system

$$\begin{cases} \dot{x}_1(t) = x_2(t) \\ \dot{x}_2(t) = -x_1(t) \end{cases}$$

Initial conditions

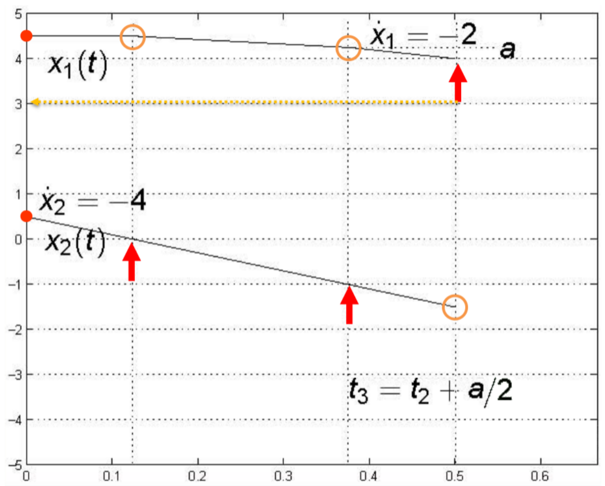
$$\begin{cases} x_1(0) = 4.5 \\ x_2(0) = 0.5 \end{cases}$$

Quantization function

$$q_i(t) = \lfloor x_i(t) \rfloor$$

Quantized system

$$\begin{cases} \dot{x}_1(t) = q_2(t) \\ \dot{x}_2(t) = -q_1(t) \end{cases}$$



QSS: motivational example (perfect oscillator)

Original system

$$\begin{cases} \dot{x}_1(t) = x_2(t) \\ \dot{x}_2(t) = -x_1(t) \end{cases}$$

Initial conditions

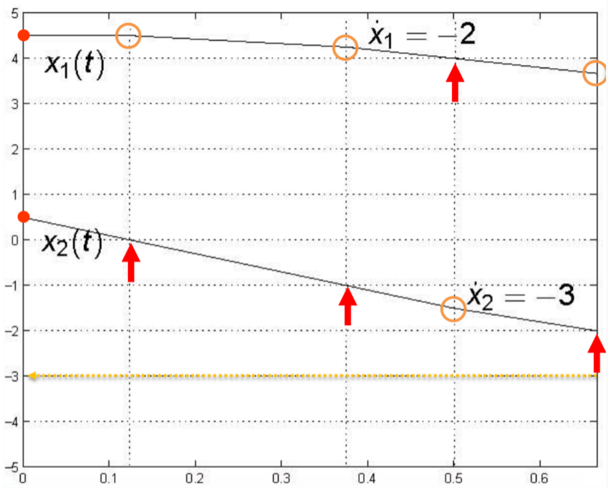
$$\begin{cases} x_1(0) = 4.5 \\ x_2(0) = 0.5 \end{cases}$$

Quantization function

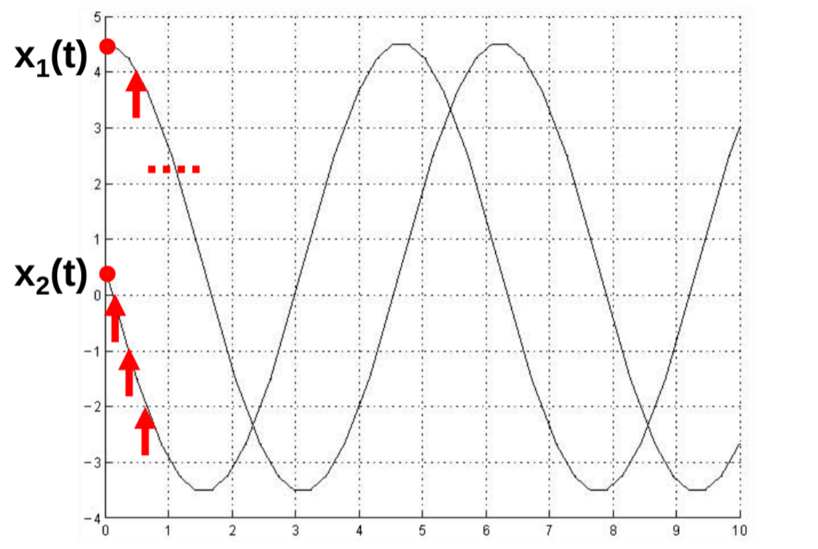
$$q_i(t) = \lfloor x_i(t) \rfloor$$

Quantized system

$$\begin{cases} \dot{x}_1(t) = q_2(t) \\ \dot{x}_2(t) = -q_1(t) \end{cases}$$

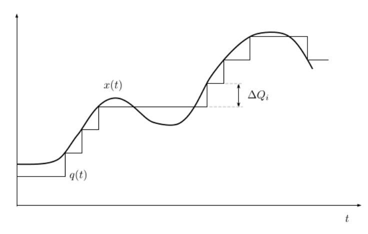


QSS: motivational example (perfect oscillator)



QSS1: first order quantization function

$$q_i(t) = \begin{cases} x_i(t) & \text{if } \left|q_i(t^-) - x_i(t)\right| \geq \Delta Q_i \\ q_i(t^-) & \text{otherwise} \end{cases}$$



- ΔQ_i is the **quantum**.
  - **Maximum deviation allowed** between x_i and q_i (error control).
  - Derived from the **precision** demanded by the user.
- Higher order methods (**QSS**n) follow essentially the same principle.
  - From the definition above, in QSS1 $q(t)$ follows piecewise constant trajectories.
  - In QSSn, $q(t)$ is composed of piecewise $(n-1)$-th order polynomials.

QSS features

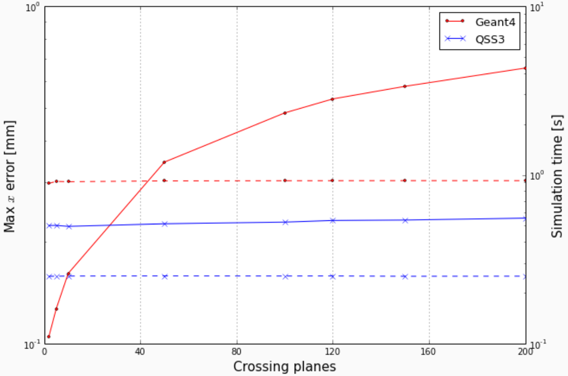
- QSS features attractive for HEP problems
  - **Asynchronicity**
    Decoupled, independent computation of changes in states variables.
  - **Lightweight discontinuity handling**
    Boundary crossings detected by solving simple (polynomial) zero-crossing functions.
  - **Dense trajectory output**
- Selected speedups reported for QSS vs. time-slicing methods in large simulation models [3][1]
  - 30x in advection reaction models (10^4 state variables)
  - 35x in logic inverters chain (10^3 state variables)
  - 100x in large spiking neurons models (4000 neurons, 80 connections per neuron)
  - 100x to 1000x in cellular division models (100 cells, 600 state variables)

Standalone tool: QSS Solver

- The QSS Solver [2] is an open-source standalone simulation tool.
- Provides C implementations for several QSS methods.
- Provides also implementations of some traditional algorithms (e.g., Dormand-Prince method).
- Our **GQLink interface** partially relies on the QSS Solver's simulation engine.

Preliminary comparison between Geant4 and QSS Solver

- Circular 2D particle motion, uniform magnetic field, equidistant parallel crossing planes.
    - Known exact analytic solution facilitates error analysis.
    - Physics processes turned off.



- With 200 plane crossings and a track length of 100 m, QSS Solver is 8x faster than Geant4[5].

Geant4 to QSS Link (GQLink): an implementation of QSS within Geant4

GQLink: QSS within Geant4

- GQLink is a proof-of-concept implementation of QSS in Geant4.
- It is based on:
  - Version `10.02.p01` of Geant4 (released February 26, 2016).
  - QSS Solver engine source code as of March 2016.
- Provides three new shared libraries to Geant4:
  - `libqss`: QSS core functionality.
  - `libgqlink`: interface API between Geant4 and QSS.
  - `libmodel`: model definition and structure (i.e., Lorentz equations).
- **It is not a new Geant4 stepper, but an abstract, clean, single entry point interface to the QSS Solver library.**
- QSS methods have complete control over the propagation for each Geant4 transportation step.
  - Usual accuracy parameters (e.g., **deltaOneStep**) do not affect GQLink simulations.
  - QSS manages accuracy in its own terms (through the control of the quantum $\Delta \mathrm{Q}$).

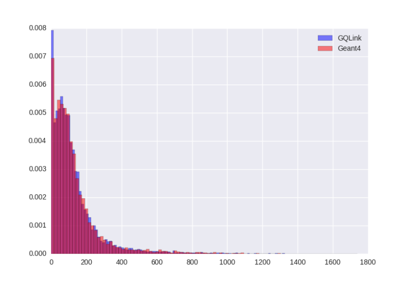
Detection of boundary crossings

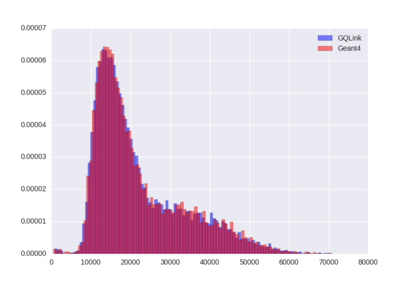
- Boundary crossings are detected through Geant4's geometry library.
- Follows same call pattern as in standard Geant4 simulations:
  - `LocateGlobalPointWithinVolume`
  - `IntersectChord`
  - `EstimateIntersectionPoint`
- **Improvement:**
  - Geant4's `AccurateAdvance` no longer used inside `EstimateIntersectionPoint`.
  - Cheaper particle transport until the crossing point (QSS polynomial dense output).
- QSS features (i.e., dense output) not fully exploited yet.

CMS application analysis

- GQLink validation was performed against a CMS application featuring:
  - Full detector geometry.
  - Volume base magnetic field.
  - Particle gun shooting π^- particles (10 GeV, 10^4 events).
  - Pythia $pp \to H \to ZZ$ (Z to all channels) ($\sqrt{s} = 14$ TeV, 50 events).

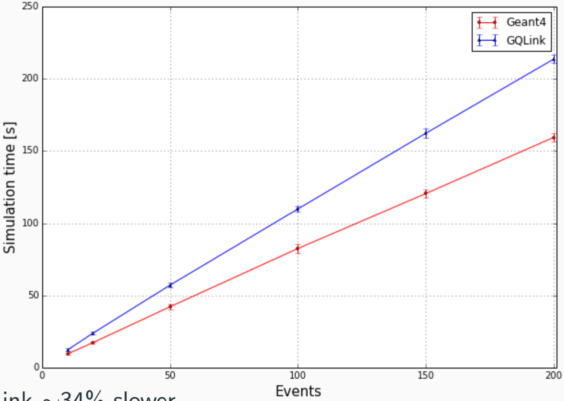
- Step count distribution for π^- (left) and secondary electrons (right) (10^4 single π^- events):





CMS application: performance comparison

- **Single π^- events**



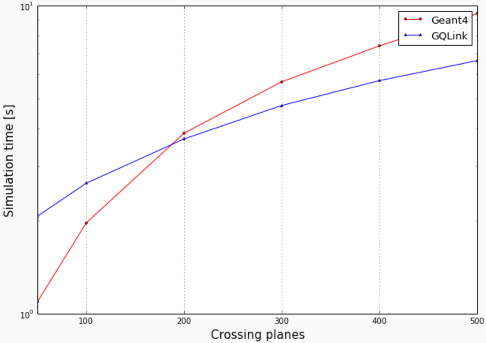
  - GQLink ~34% slower.
- **Pythia events**
  - GQLink ~49% slower (6.67 hours vs. 4.46 hours).
- Geant4 stepper: `G4ClassicalRK4` (accuracy set to $\epsilon = 10^{-5}$).

Discussion

- GQLink currently uses QSS3, a third order method, whereas Geant4 uses fourth order Runge-Kutta methods.
- The observed simulation time can be partially explained by this fact, since lower order methods typically require more computational steps to achieve the same accuracy.
- QSS4 is still experimental, but GQLink will transparently support it once it becomes available.

Alternative scenario: helix and parallel planes

- Different scenario: helix trajectory crossing parallel equidistant planes & more frequent boundary crossings.
- No stepwise abrupt changes in the direction/velocity of the particle.
  - i.e., physics processes turned off.
- Using `G4ClassicalRK4` stepper (accuracy set to $\epsilon = 10^{-5}$).



- GQLink outperforms Geant4 when using ≥ 200 planes ($\sim$42% faster for 500 planes).

Conclusions and future work

Conclusions

- We developed **GQLink**, a prototype for QSS methods within Geant4.
- Validation: number of steps and tracks produced are statistically consistent with Geant4's for both toy examples and realistic HEP applications.
- Performance:
  - We found that GQLink can outperform Geant4 on certain simplified scenarios.
  - Preliminary tests revealed GQLink is currently $\sim$34% slower than standard Geant4 in a full CMS realistic scenario (using single π^- events).
- We aim at performing with GQLink, automatically:
  - At least no worse than standard Geant4 in the general case.
  - Much better than Geant4 in those cases that leverage combinations of QSS features (scenario–dependent, e.g. intense boundary crossing).
- From an abstract viewpoint, GQLink also opens new possibilities to interface Geant4 with any external stepper.

Future work

- Exploit fully the QSS capabilities for efficient geometry crossing detection.
- Improve the performance of QSS for the reinitialization of momentum variables forced from Geant4 upon starting a new step.

Acknowledgments

- Rodrigo Castro and Nicolás Ponieman (QSS team at UBA-CONICET)
- Federico Bergero, Joaquín Fernández and Ernesto Kofman (QSS team at UNR-CONICET)
- Soon Yung Jun, Krzysztof Genser and Daniel Elvira (FNAL)

Thank you!

Questions?

Backup slides

QSS: definition

- Consider the initial-value problem

$$\begin{cases} \dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t)) \\ \mathbf{x}(t_0) = \mathbf{x}_0 \end{cases}$$

- QSS simulates the following approximate system,

$$\begin{cases} \dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{q}(t)) \\ \mathbf{q}(t_0) = \mathbf{x}_0 \end{cases}$$

where $\mathbf{x}(t)$ and $\mathbf{q}(t)$ are related by a quantization function.

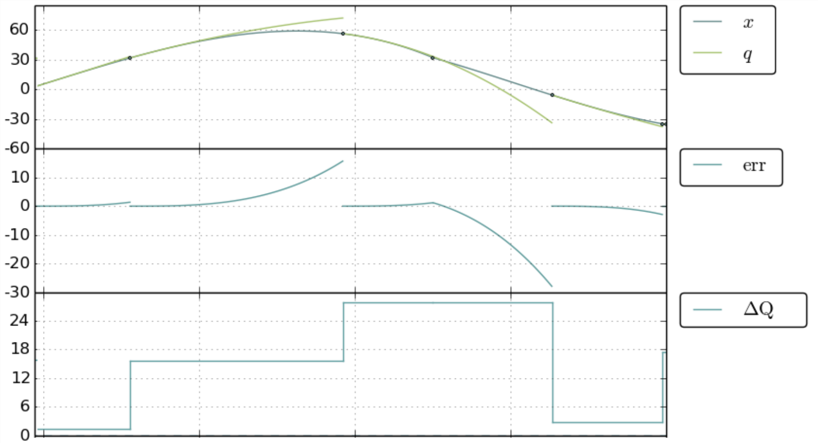
QSS: features of $x(t)$

- In QSS1, $q(t)$ follows piecewise constant trajectories, and we have that
$$\dot{x}(t) = f(q(t))$$
$\Rightarrow x(t)$ follows piecewise linear trajectories.
- However, in higher order QSS methods we cannot derive similar conclusions for an arbitrary nonlinear f.
- To overcome this, QSSn approximates f through its Taylor expansion up to the n-th term.
  - In general, derivatives of f are computed numerically.
- Using this, it can be seen that $x(t)$ is composed of piecewise n-th order polynomials in QSSn.

QSS: integration steps

- An integration step occurs when the difference between $q(t)$ and its related $x(t)$ equals the quantum, $\Delta\mathrm{Q}$.
  - $q(t)$ needs to be recomputed using the quantization function.
  - Any other state variable whose derivative depends on $q(t)$ has to be reevaluated too.
  - Finally, quantization times (i.e., times of the upcoming integration steps) are updated using these new polynomial expressions (by computing polynomial roots).

QSS3 sample plot



Quantization in QSS3

- Suppose an integration step on time t_j.
- $q(t)$ is recomputed using the Taylor expansion of $x(t)$ up to its third term:

$$x(t) \approx \underbrace{x(t_j) + \dot{x}(t_j)\,(t - t_j) + \frac{\ddot{x}(t_j)}{2}\,(t - t_j)^2}_{q(t)}$$

- In QSS3, $x(t)$ is composed of piecewise cubic polynomials. Then,

$$x(t) = a_0 + a_1\,(t - t_k) + a_2\,(t - t_k)^2 + a_3\,(t - t_k)^3$$

  where $t_k < t_j$ is last time on which $x(t)$ was updated.
- Thus,
  - $\dot{x}(t_j) = a_1 + 2a_2\,(t_j - t_k) + 3a_3\,(t_j - t_k)^2$
  - $\ddot{x}(t_j) = 2a_2 + 6a_3(t_j - t_k)$

Quantized approximation of Lorentz equations

Lorentz equations

$$\begin{cases} \dot{x} = v_x & \dot{v}_x = \frac{q\,c^2}{m\,\gamma} \cdot (v_y\,B_z - v_z\,B_y) \\ \dot{y} = v_y & \dot{v}_y = \frac{q\,c^2}{m\,\gamma} \cdot (v_z\,B_x - v_x\,B_z) \\ \dot{z} = v_z & \dot{v}_z = \frac{q\,c^2}{m\,\gamma} \cdot (v_x\,B_y - v_y\,B_x) \end{cases}$$

- x, y, z, v_x, v_y, v_z are the state variables

$$\Downarrow$$

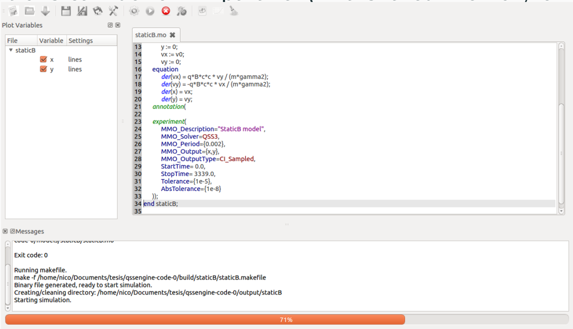
Quantized approximation

$$\begin{cases} \dot{x} = q_{v_x} & \dot{v}_x = \frac{q\,c^2}{m\,\gamma} \cdot (q_{v_y}\,B_z - q_{v_z}\,B_y) \\ \dot{y} = q_{v_y} & \dot{v}_y = \frac{q\,c^2}{m\,\gamma} \cdot (q_{v_z}\,B_x - q_{v_x}\,B_z) \\ \dot{z} = q_{v_z} & \dot{v}_z = \frac{q\,c^2}{m\,\gamma} \cdot (q_{v_x}\,B_y - q_{v_y}\,B_x) \end{cases}$$

- Each state variable s is approximated by the quantized variable q_s

QSS Solver

- Models are written in the μ-Modelica language (more detail next).
- Offers native implementations of:
  - Standard QSS methods (QSS1, QSS2, QSS3, QSS4[1])
  - A QSS method for marginally stable systems (CQSS)
  - QSS methods for stiff systems (LIQSS1, LIQSS2, LIQSS3)
  - Dormand-Prince method (DOPRI5)
  - Differential/Algebraic System Solver method (DASSL)
- Simulations can be run in parallel (in a shared memory environment).



[1] Still experimental

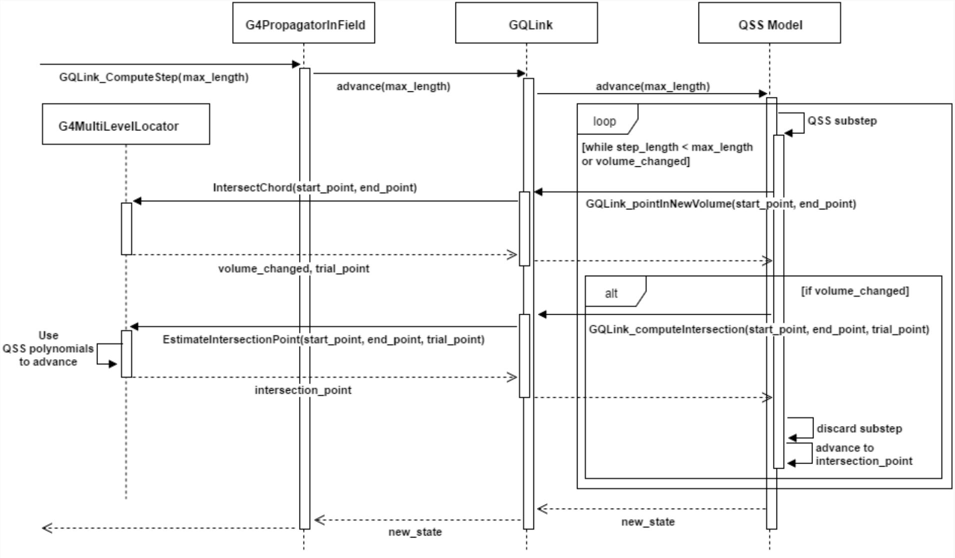
μ-Modelica language

- Subset of the standard Modelica modeling language.
  - Free, high-level, object-oriented language for modeling of large, complex, and heterogeneous systems.
- Models are mathematically described by differential, algebraic and discrete equations.

Example: Lorentz equations in GQLink

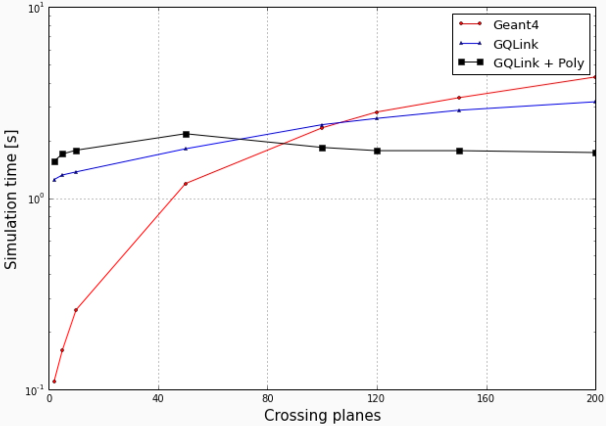
```
Bx = GQLink_GetBx(x,y,z);
By = GQLink_GetBy(x,y,z);
Bz = GQLink_GetBz(x,y,z);

der(x)  = vx;
der(y)  = vy;
der(z)  = vz;
der(vx) = (q*c*c / (m*gamma)) * (Bz*vy - By*vz);
der(vy) = (q*c*c / (m*gamma)) * (Bx*vz - Bz*vx);
der(vz) = (q*c*c / (m*gamma)) * (By*vx - Bx*vy);
```

Stepping in GQLink: sequence diagram

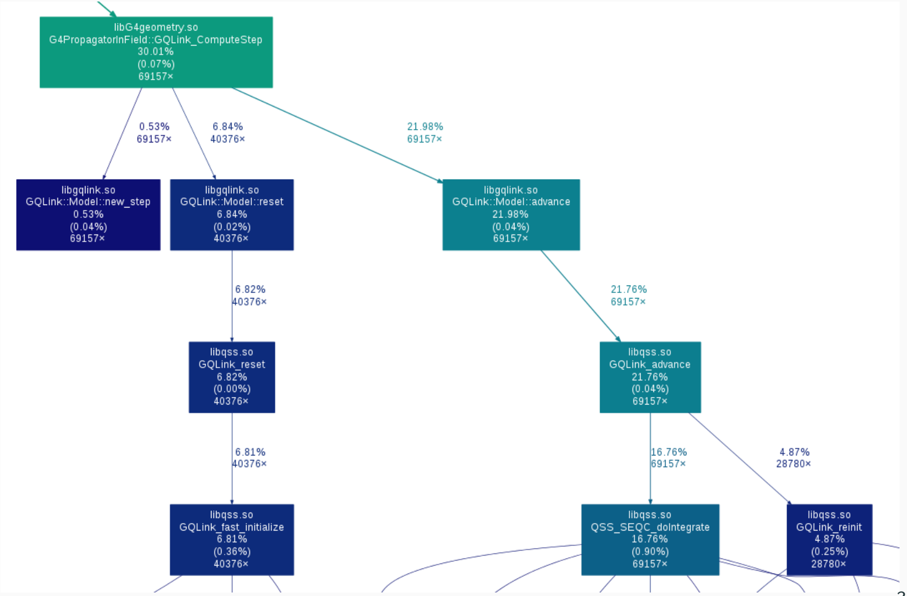


Boundary detection through QSS polynomials

- We developed a prototype of boundary detection using QSS polynomials for regular boxes.
- Computation of a polynomial root instantly yields the time at which the boundary will be crossed.



Profiling of GQLink_ComputeStep



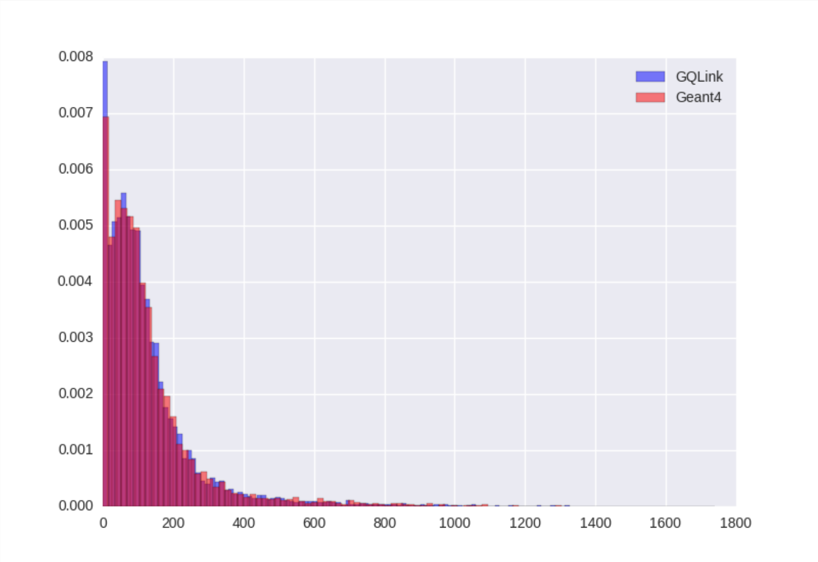


Figure 1: π^- steps

Step count distribution (10^4 single π^- events) - 2/7

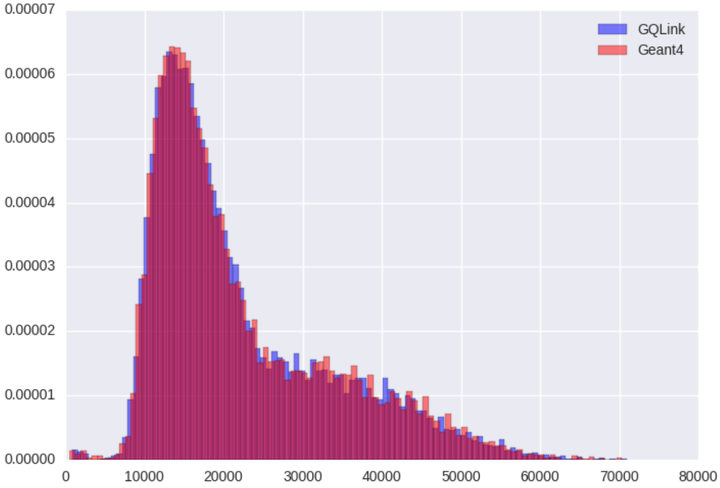


Figure 2: secondary electron steps

Step count distribution (10^4 single π^- events) - 3/7

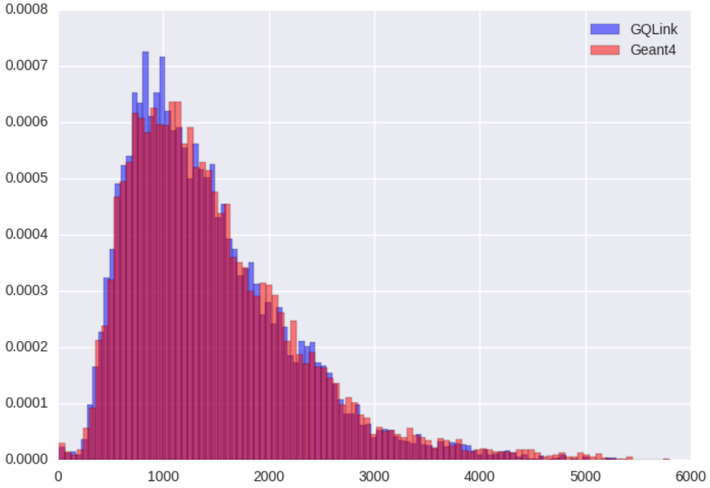


Figure 3: secondary e^+ steps

Step count distribution (10^4 single π^- events) - 4/7

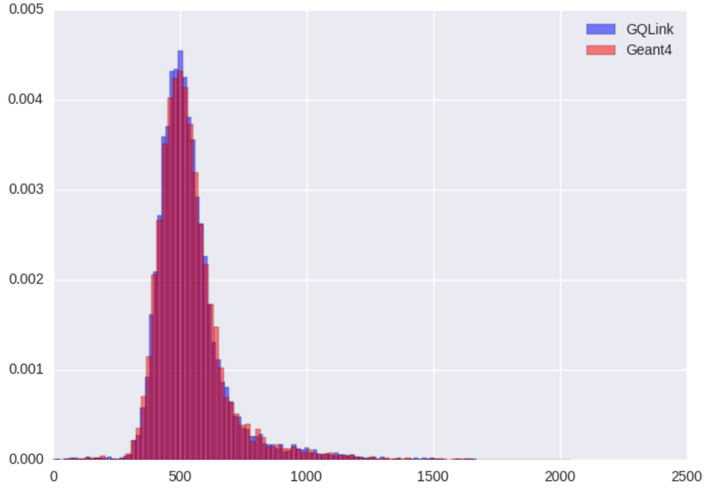


Figure 4: secondary π^+ steps

Step count distribution (10^4 single π^- events) - 5/7

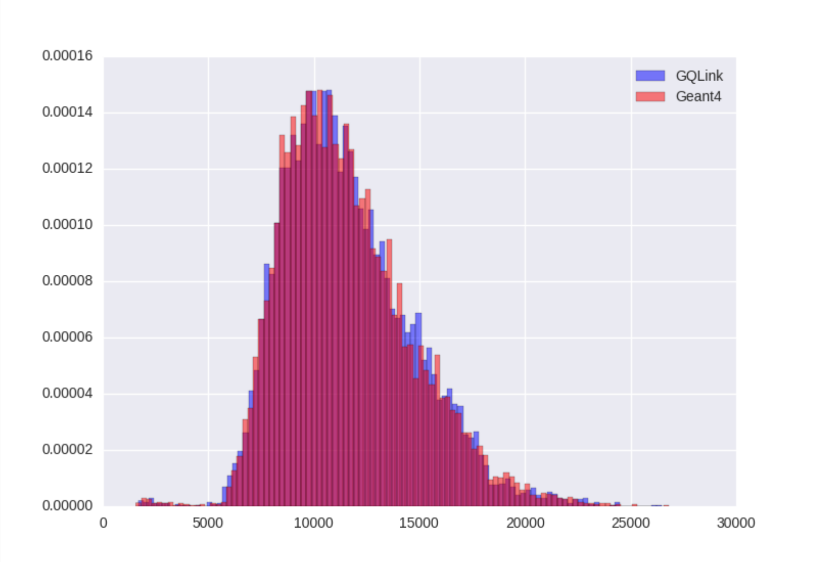


Figure 5: secondary γ steps

Step count distribution (10^4 single π^- events) - 6/7

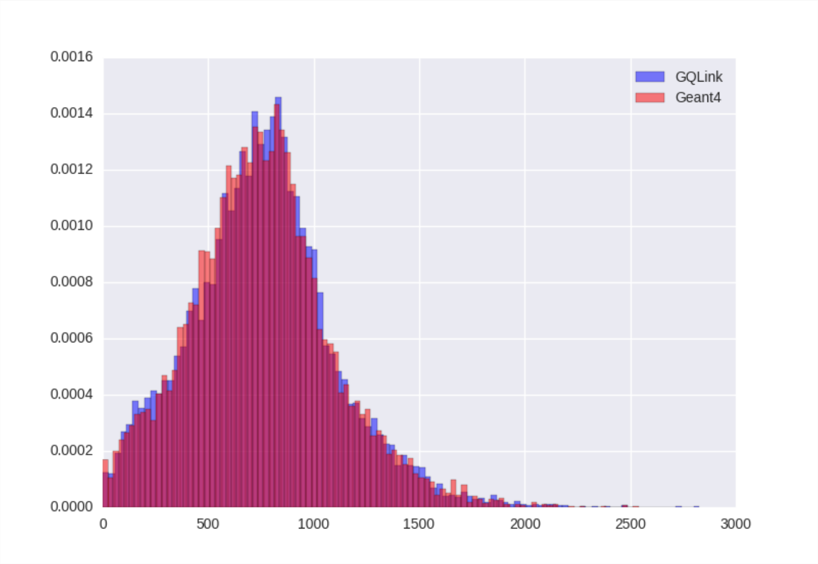


Figure 6: secondary proton steps

Step count distribution (10^4 single π^- events) - 7/7

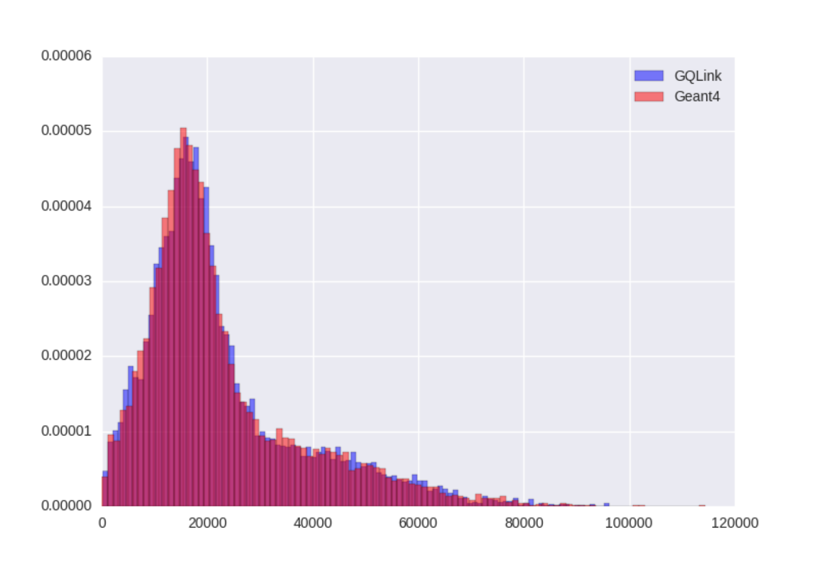


Figure 7: other secondaries steps

References I

F. Bergero, J. Fernndez, E. Kofman, and M. Portapila.
Quantized State Simulation of Advection–Diffusion–Reaction Equations.
In *Mecánica Computacional*, volume XXXII, pages 1103–1119, Mendoza, Argentina, 2013. Asociación Argentina de Mecánica Computacional.

J. Fernández and E. Kofman.
A Stand–Alone Quantized State System Solver. Part I.
In *Proc. of RPIC 2013*, Bariloche, Argentina, 2013.

G. Grinblat, H. Ahumada, and E. Kofman.
Quantized State Simulation of Spiking Neural Networks.
Simulation: Transactions of the Society for Modeling and Simulation International, 88(3):299–313, 2012.

References II

E. Kofman and S. Junco.
Quantized State Systems. A DEVS Approach for Continuous System Simulation.
Transactions of SCS, 18(3):123–132, 2001.

N. Ponieman.
Aplicación de métodos de integración por cuantificación al simulador de partículas geant4.
Master's thesis, Facultad de Ciencias Exactas y Naturales. Universidad de Buenos Aires., 2015.