Charge Transport Methods

Koen Wolters - *ETH Zurich*
Allpix Squared User Workshop

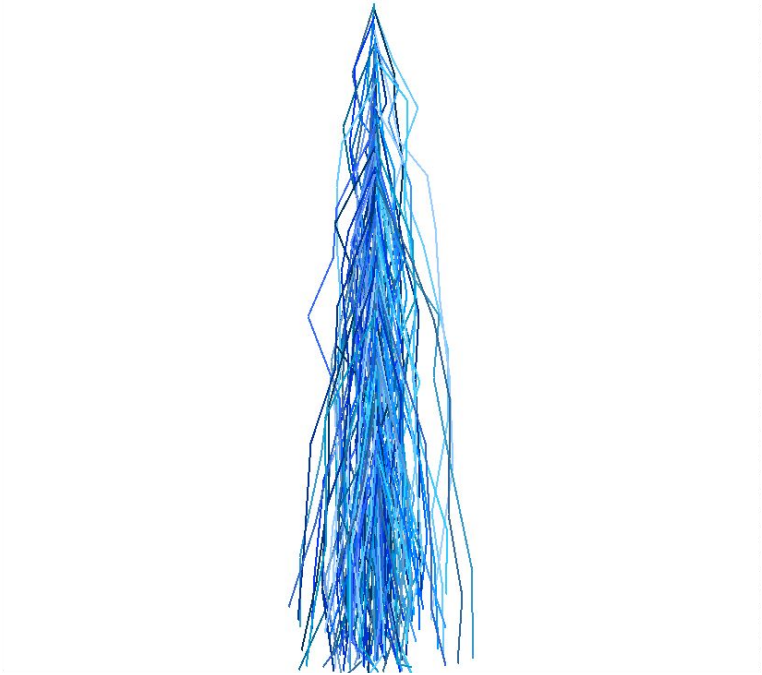
27 November 2018

About me

- Currently pursuing a master degree at ETH Zurich
- Involved in Allpix Squared development from the beginning
- Developed substantial part of the framework as Technical Student from February - August 2017
  - Designed overall framework architecture
  - Implemented initial modules (ported basics from AllPix)
- Continued participation in development (mostly as reviewer)

Outline

- Introduction
- Modules and configurations
- Visualization
- Simulation flows
- **Module modifications**
- Conclusion



Focus on framework structure and possibilities, more complete (realistic) simulations in other presentations

Will describe only use the **master** *version*[†]

[†] Actually not because I found a few minor bugs for which I opened a PR, but this has not been merged yet

Introduction

Before propagation

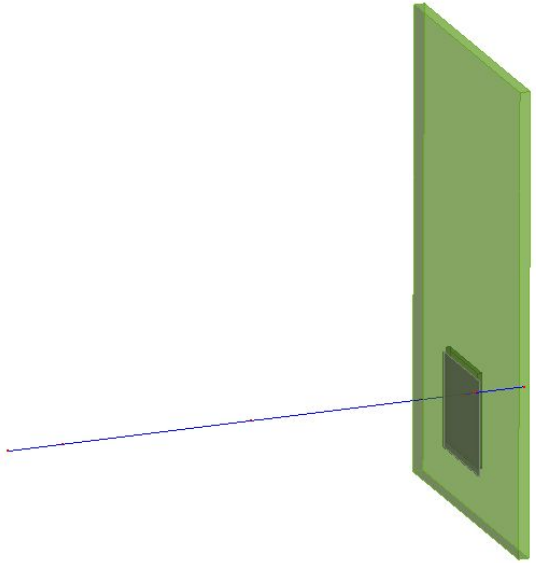
- Use setup roughly based on reference paper[†] first (*more simulation flows later)*
- Setup basic framework configuration
- Define detector setup *(next page)*
- Deposit charges in the sensor using Geant4
- Read simple linear electric field

```
[Allpix]
log_level = "WARNING"
log_format = "DEFAULT"
detectors_file = "workshop-geometry.conf"
number_of_events = 10000
random_seed = 0

[GeometryBuilderGeant4]

[DepositionGeant4]
physics_list = FTFP_BERT_EMY
Enable_pai = true
particle_type = "pi+"
source_energy = 120GeV
source_position = 0 0 -10mm
source_type = "beam"
beam_size = 2mm
beam_direction = 0 0 1
number_of_particles = 1
max_step_length = 1um

[ElectricFieldReader]
name = "dut"
model = "linear"
depletion_voltage = -7V
bias_voltage = -15V
```

[†]S. Spannagel et al., **Allpix2: A modular simulation framework for silicon detectors**, Nucl. Instr. Meth. A 901 (2018) 164 – 172, doi:10.1016/j.nima.2018.06.020, arXiv:1806.05813

Detector setup

One single 100µm thick Timepix DUT



```
[dut]
type = "timepix"
sensor_thickness = 100um
position = 0mm 0mm 0mm
orientation = 0deg 0deg 180deg
alignment_precision_position = 1mm
1mm 100um
alignment_precision_orientation =
0.2deg 0.2deg 0.2deg
```

Goal of propagator module

DepositedCharge

- ★ Global position
- ★ Local position
- ★ Carrier type (electron/hole)
- ★ Charge count
- ★ Deposition time†
- ★ Monte-Carlo particle (producer)

PropagatedCharge

- ★ Global position
- ★ Local position
- ★ Carrier type (electron/hole)
- ★ Charge count
- ★ Propagation finish time
- ★ Deposited charge *(producer)*

Output from deposition module (Geant4) bound by propagator

Partial propagated charges are not dropped, up to transfer module to determine charges to transfer

† [bug] https://gitlab.cern.ch/allpix-squared/allpix-squared/issues/138

Propagation

- Charge propagation: core component for pixel detectors simulation
- Monte-Carlo approach for simulating (millions of) events
- Simulation not as detailed as in TCAD as it is infeasible (hours per event)
- Drift-diffusion model for single sets of carriers
    - Drift: drift velocity in static electric and magnetic field using field-dependent mobility parameterization (*more about TCAD fields and mobility models in other presentations)*
    - Diffusion: random walk of charge carriers influenced by mobility model

Simulation speed

- Fast computation per event (compared to TCAD), many events remain computation-intensive ➔ propagation often bottleneck

  |15:07:57.397| (STATUS) Executed 7 instantiations in 2 minutes 12 seconds, spending 76% of time in slowest instantiation GenericPropagation:dut

  10000 events
- Balance accuracy and speed of simulation
  - Choose number of events to simulate
  - Selecting appropriate module
  - Adapting module configuration
  - Different configurations for the simulation chain
  - Use a powerful computer and enable multithreading (when setup allows)

Simulation speed

- Fast computation per event (compared to TCAD), many events remain computation-intensive ➔ propagation often bottleneck

  |15:07:57.397| (STATUS) Executed 7 instantiations in 2 minutes 12 seconds, spending 76% of time in slowest instantiation GenericPropagation:dut

  10000 events
- Balance accuracy and speed of simulation
  - Choose number of events to simulate
  - **Selecting appropriate module**
  - Adapting module configuration
  - Different configurations for the simulation chain
  - Use a powerful computer and enable multithreading (when setup allows)
- Two propagation modules
  - ProjectionPropagation: direct projection of charges under constraints
  - GenericPropagation: generic step-by-step propagator

ProjectionPropagation module

Charge projection

- Constraining conditions simplifies calculations significantly
  - Use linear electric field
  - No magnetic field
  - Simplify (Jacobini) mobility parameterization ($\beta = 1$, acceptable around room temperature)

Charge projection

- Constraining conditions simplifies calculations significantly
  - Use linear electric field
  - No magnetic field
  - Simplify (Jacobini) mobility parameterization ($\beta = 1$, acceptable under certain conditions)
- Without diffusion final position is direct projection on sensor surface
- Drift time can be estimated with an analytical integral

$$t = \int \frac{1}{v} ds = \int \frac{1}{\mu(s) E(s)} ds = \int \frac{\left(1 + \left(\frac{E(S)}{E_c}\right)^{\beta}\right)^{1/\beta}}{\mu_0 E(s)} ds \approx \frac{1}{\mu_0} \int \left(\frac{1}{ks + E_0} + \frac{1}{E_c}\right) ds \quad \rightarrow \quad t = \frac{1}{\mu_0} \left[\frac{\ln(ks + E_0)}{k} + \frac{s}{E_c}\right]_a^b$$

- Use estimated drift-time to approximate diffusion standard deviation, then draw total offset from normal distribution

Options

Simulation results with different settings

Baseline configuration

```
[ProjectionPropagation]
name = "dut"
temperature = 293K
charge_per_step = 50
```

Options

Simulation results with different settings

- **temperature**

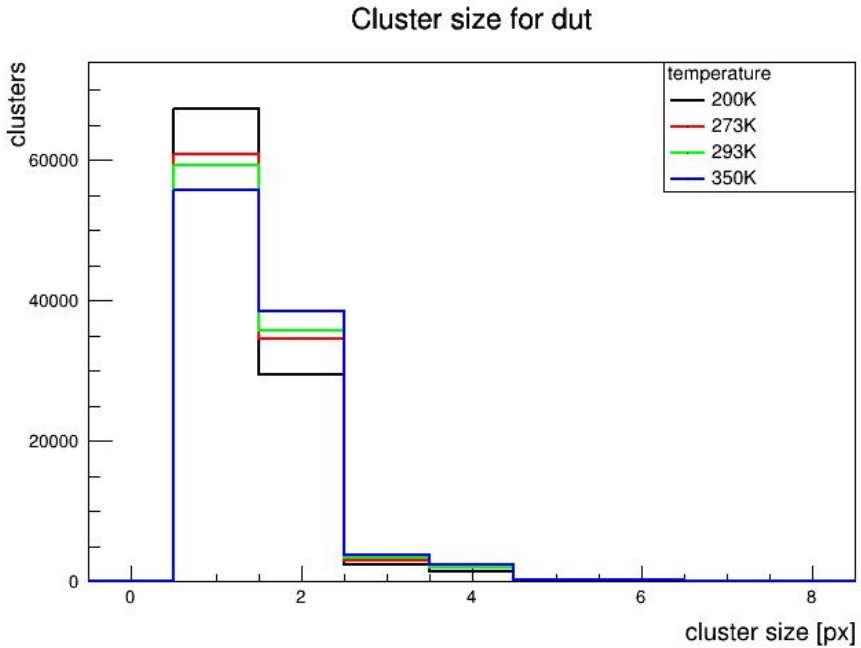
Influences drift time estimation error!

200K	*273K*	*293K*	*350K*
?	?	?	?

Baseline configuration

```
[ProjectionPropagation]
name = "dut"
temperature = 293K
charge_per_step = 50
```

10000 events



Options

Simulation results with different settings

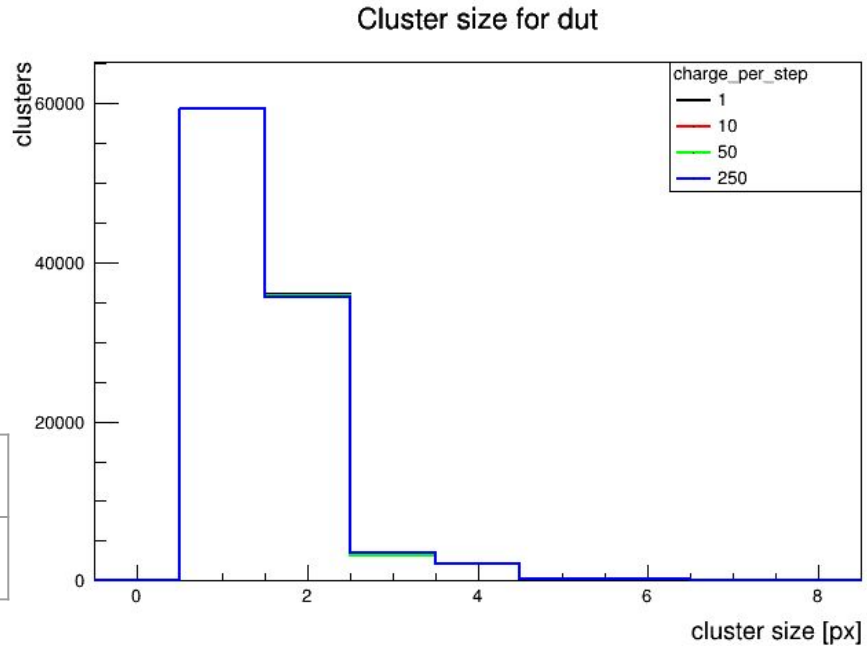
- temperature
- **charge_per_step**

Maximum charge per step is bounded by deposition max_step_length

Baseline configuration

```
[ProjectionPropagation]
name = "dut"
temperature = 293K
charge_per_step = 50
```

100000 events



1	*10*	*50*	*250*
4676s	674s	296s	241s

Options

Simulation results with different settings

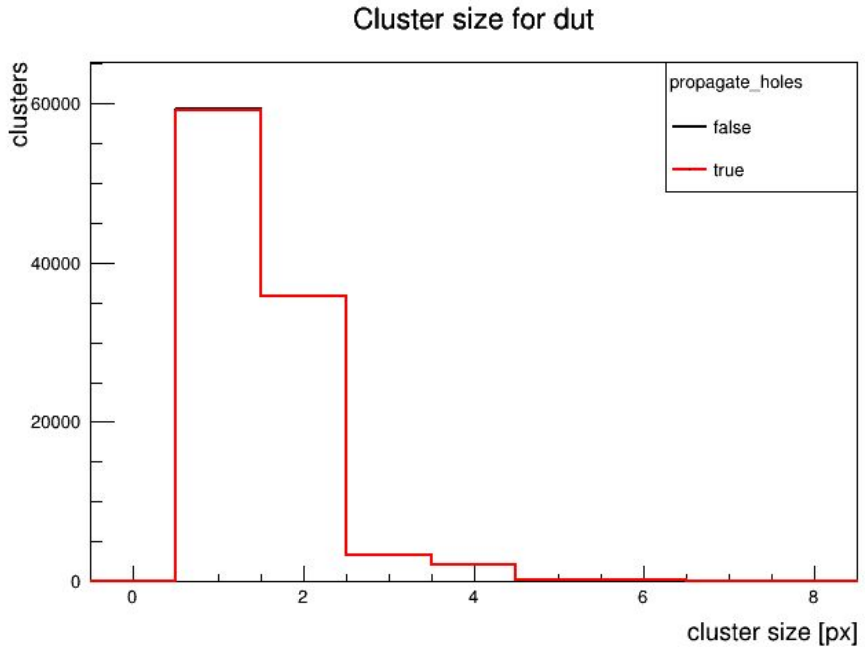
- temperature
- charge_per_step
- **propagate_holes**

Electric field has to be inverted

Baseline configuration

```
[ProjectionPropagation]
name = "dut"
temperature = 293K
charge_per_step = 50
```

100000 events



false	*true*
254s	252s

Options

Simulation results with different settings

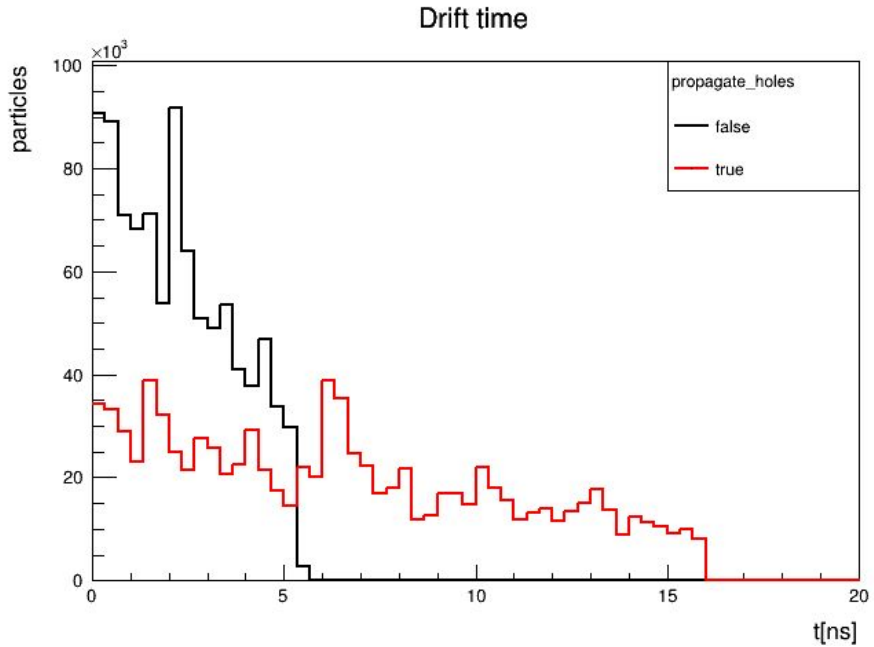
- temperature
- charge_per_step
- **propagate_holes**

Electric field has to be inverted

Baseline configuration

```
[ProjectionPropagation]
name = "dut"
temperature = 293K
charge_per_step = 50
```

100000 events



false	*true*
254s	252s

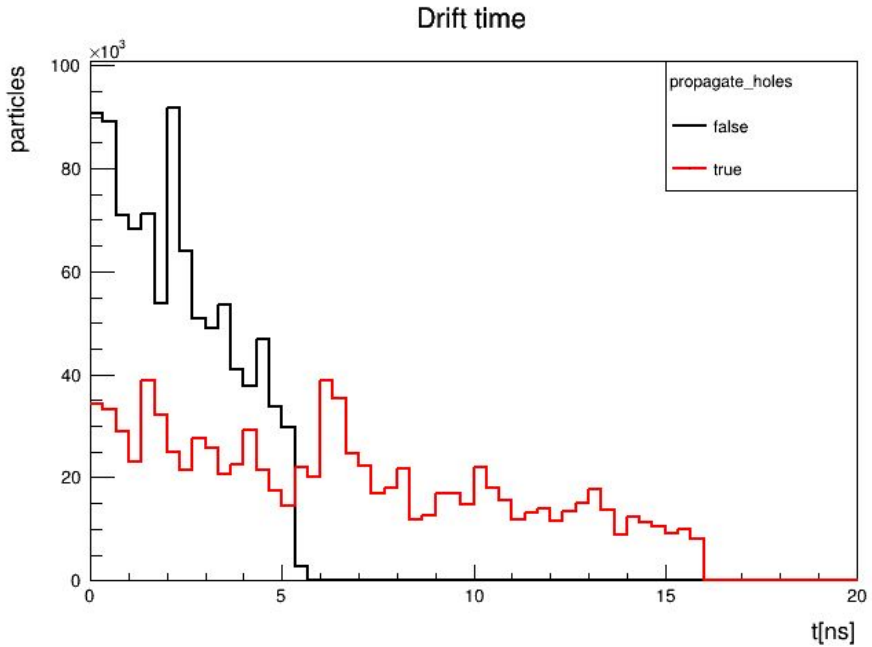
Options

Simulation results with different settings

- temperature
- charge_per_step
- propagate_holes
- ***ignore_magnetic_field***
- **output_plots**

Baseline configuration

```
[ProjectionPropagation]
name = "dut"
temperature = 293K
charge_per_step = 50
```

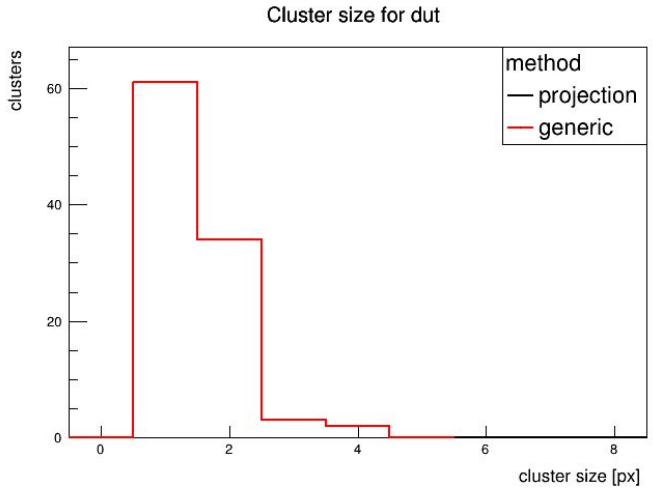


false	*true*
251s	253s

Comparison

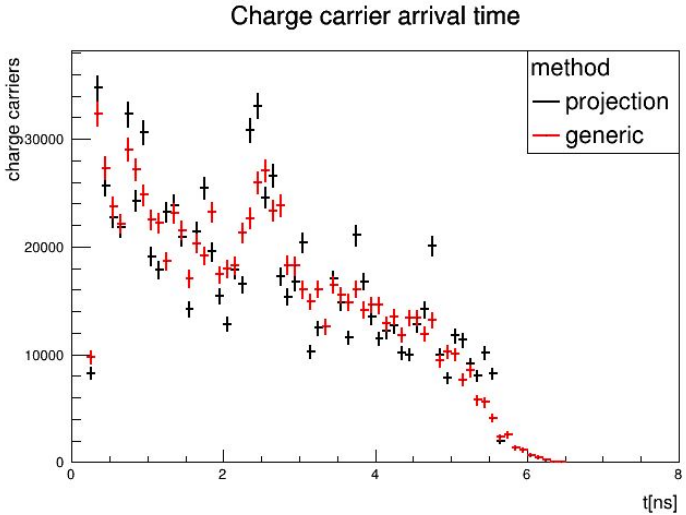
- Comparison between projection propagation and generic propagation
- Simple example without imported electric or magnetic field

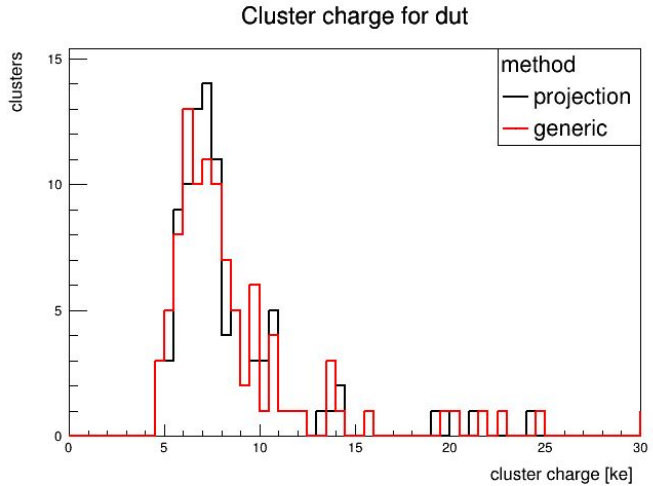
method	**time**
projection	?
generic	?



Comparison

- Comparison between projection propagation and generic propagation
- Simple example without imported electric or magnetic field





GenericPropagation module

Generic propagation

- Workhorse of the framework, provides generic propagation with support for
  - Custom electric fields imported from TCAD (*see talk by Magdalena*)
  - Custom magnetic fields for simulating Lorentz drift (*see talk by Paul*)
  - Random offset per step to simulate diffusion
- Powered by built-in 4th-order Runge-Kutta solver with 5th-order error estimation to determine appropriate step size
- Propagation performed until configured integration time is reached
- Optimized for speed, while having simple configuration, allowing for several visualization possibilities *(more later)* and keeping code clean to make modifications easy *(more later)*

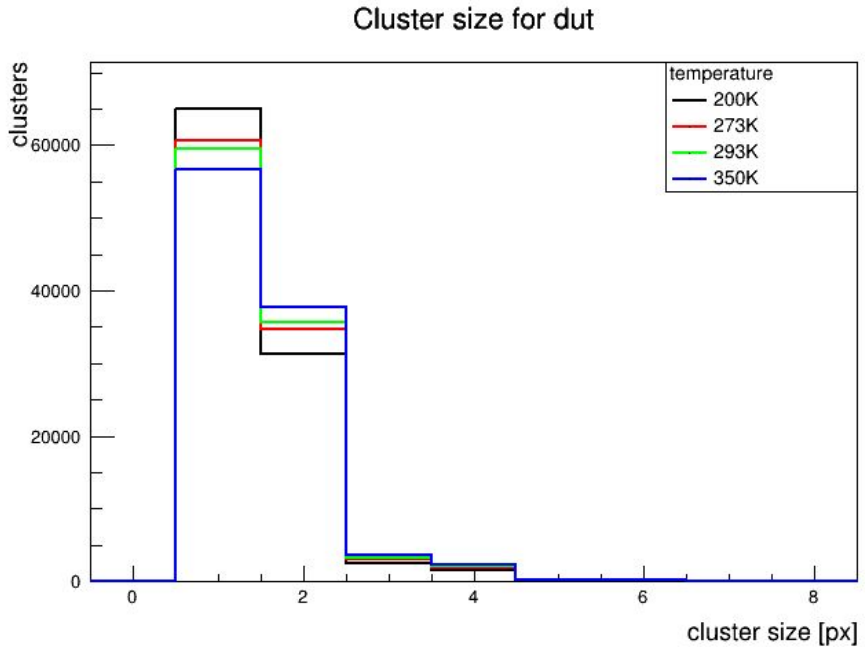
Generic options

Simulation results with different settings

- **temperature**

Baseline configuration

```
[GenericPropagation]
name = "dut"
temperature = 293K
charge_per_step = 100
integration_time = 100ns
```



200K	*273K*	*293K*	*350K*
540s	594s	608s	657s

Generic options

Simulation results with different settings

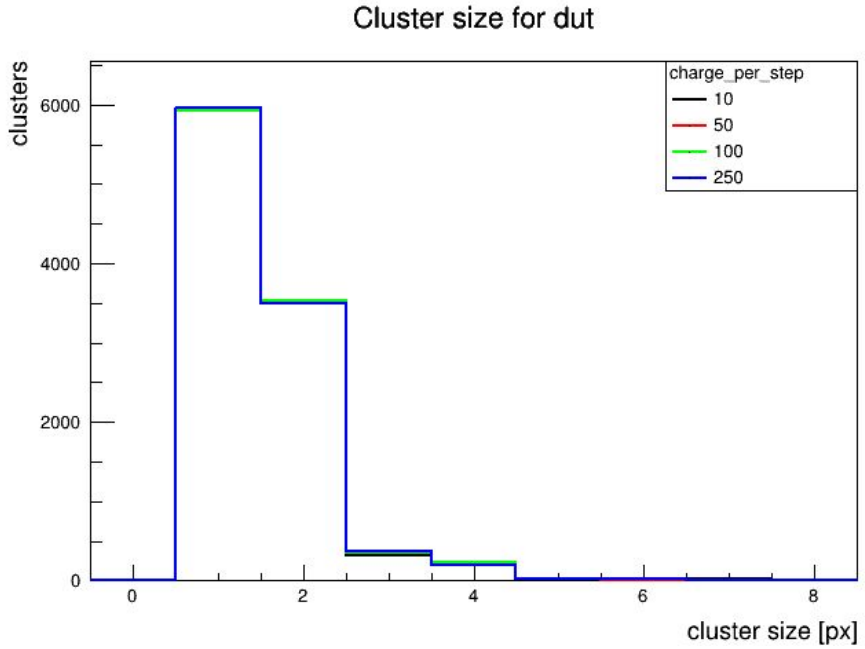
- temperature
- **charge_per_step**

Maximum charge per step is bounded by deposition max_step_length

Baseline configuration

```
[GenericPropagation]
name = "dut"
temperature = 293K
charge_per_step = 100
integration_time = 100ns
```

10000 events



10	*50*	*100*	*250*
332s	95s	66s	50s

Generic options

Simulation results with different settings

- temperature
- charge_per_step
- **propagate_electrons**
- **propagate_holes**

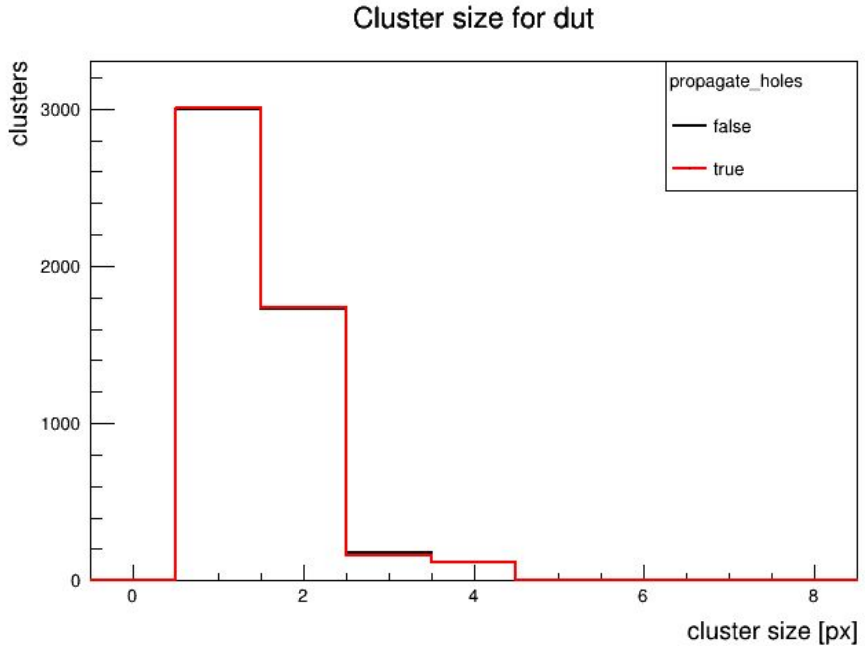
Electric field needs to be inverted for holes instead of electrons. Enable both (now only) for visualization

false	*true*
69s	89s

Baseline configuration

```
[GenericPropagation]
name = "dut"
temperature = 293K
charge_per_step = 100
integration_time = 100ns
```

10000 events



Generic options

Simulation results with different settings

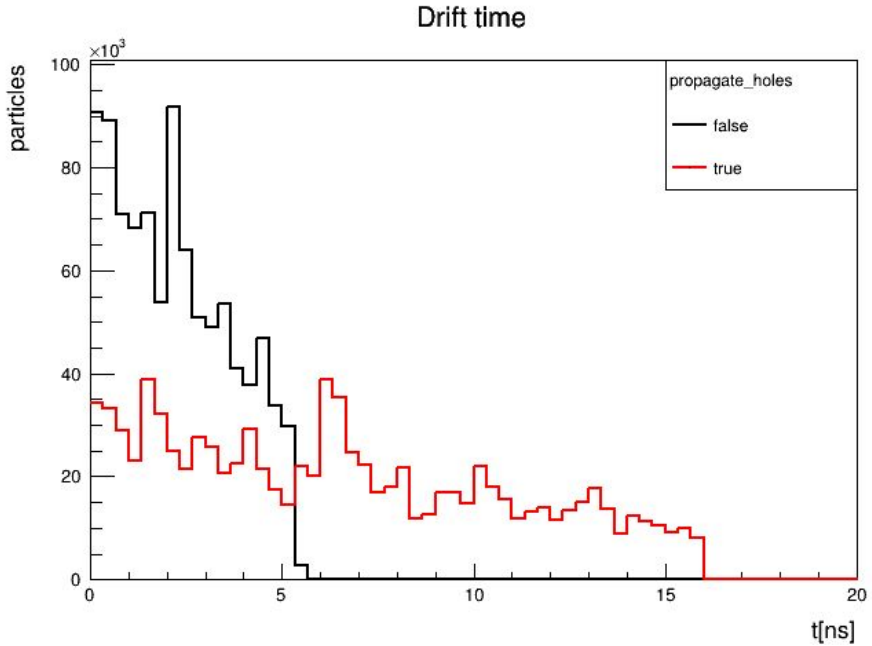
- temperature
- charge_per_step
- propagate_electrons
- propagate_holes
- ***ignore_magnetic_field***
- **output_plots**

false	*true*
11s	28s

Baseline configuration

```
[GenericPropagation]
name = "dut"
temperature = 293K
charge_per_step = 100
integration_time = 100ns
```

1000 events



Specific options for integration

Simulation results with different settings

- **spatial_precision**

Not length of step, but estimated error between exact and numerical integral!

Estimated only for drift!

0.25nm
66s

Baseline configuration

```
[GenericPropagation]
name = "dut"
temperature = 293K
charge_per_step = 100
integration_time = 100ns
```

Simple integration:
estimated precision always < 1 pm outside the neighborhood of the sensor edge with linear electric field and no magnetic field

Specific options for integration

Simulation results with different settings

- spatial_precision
- **integration_time**

Important with non-depleted regions with more significant diffusion effects

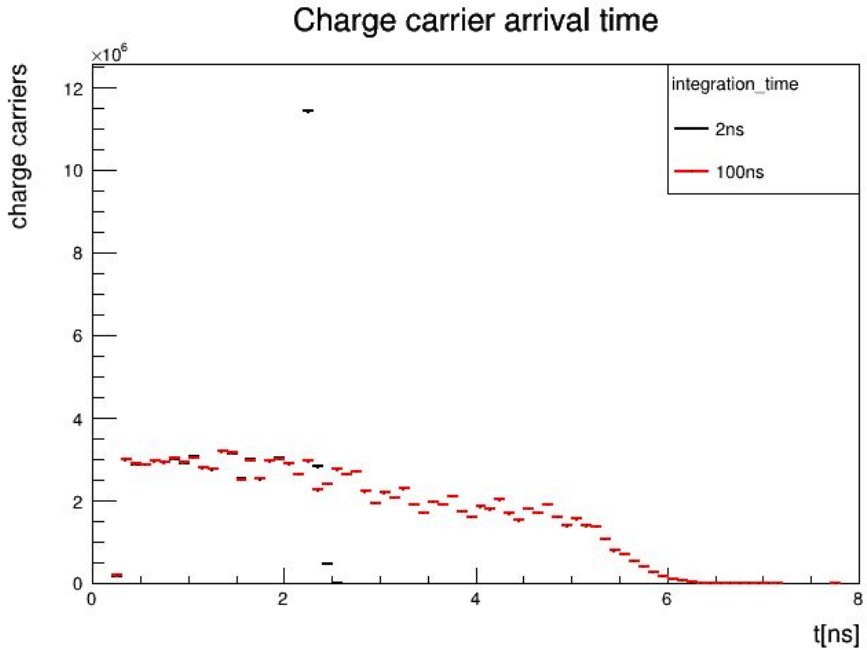
Here all carriers reach the implant…

2ns	*100ns*
58s	66s

Baseline configuration

```
[GenericPropagation]
name = "dut"
temperature = 293K
charge_per_step = 100
integration_time = 100ns
```

10000 events



Specific options for integration

Simulation results with different settings

- spatial_precision
- integration_time
- **timestep_start**

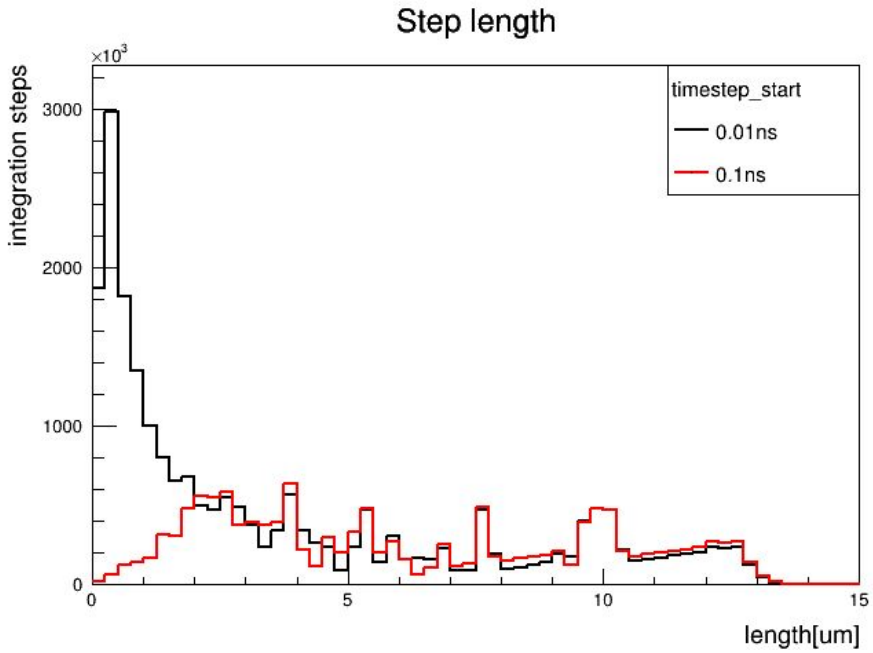
Important as step size is only updated for precision in next step!

0.01ns	*0.1ns*
69s	55s

Baseline configuration

```
[GenericPropagation]
name = "dut"
temperature = 293K
charge_per_step = 100
integration_time = 100ns
```

10000 events



Specific options for integration

Simulation results with different settings

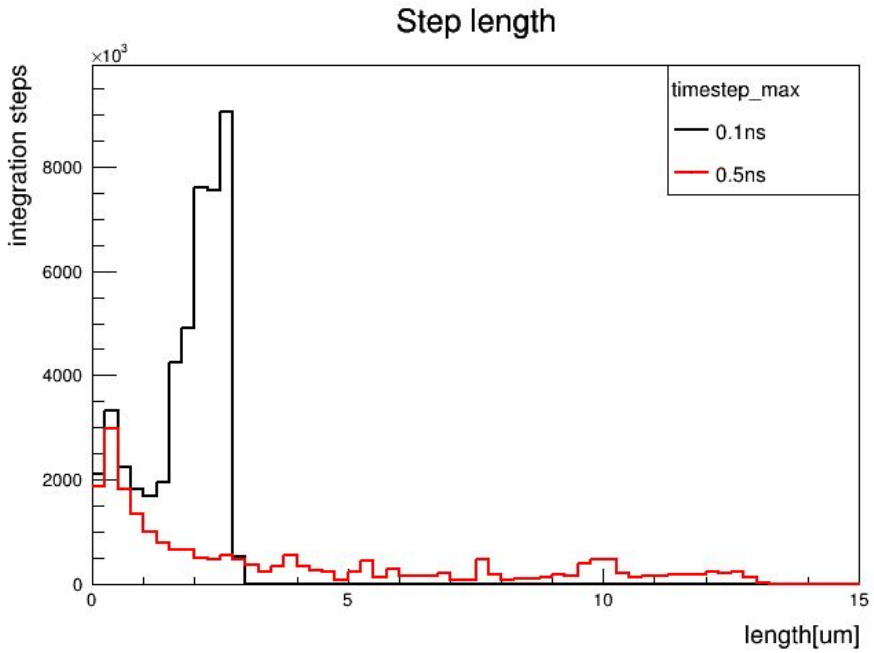
- spatial_precision
- integration_time
- timestep_start
- **timestep_max**

Useful to ensure precise steps in visualization and boundary behavior

0.1ns	*0.5ns*
110s	71s

Baseline configuration

[GenericPropagation]
name = "dut"
temperature = 293K
charge_per_step = 100
integration_time = 100ns



Specific options for integration

Simulation results with different settings

- spatial_precision
- integration_time
- timestep_start
- timestep_min
- **timestep_min**

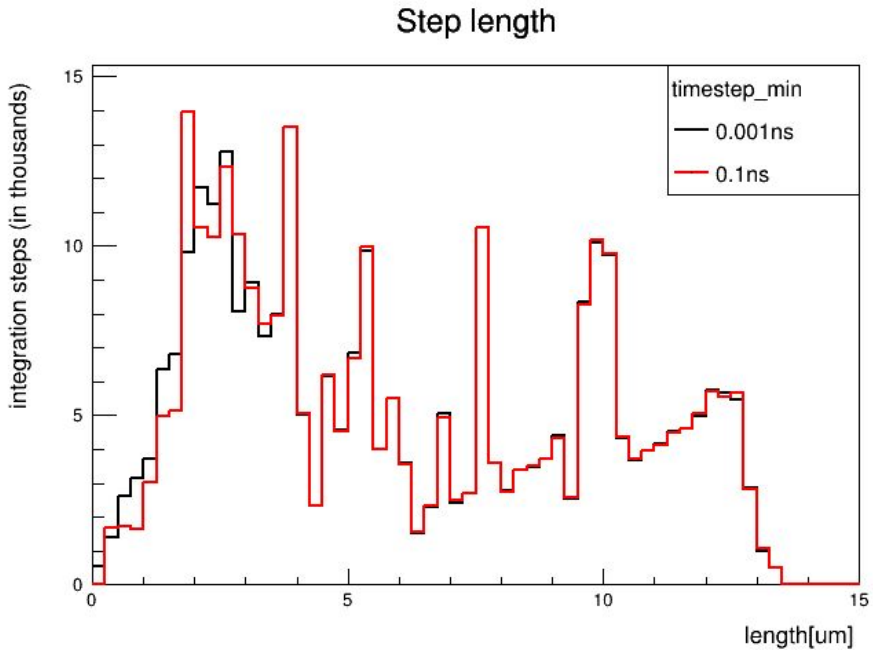
Useful to prevent too slow integration

0.001ns	*0.1ns*
?	?

timestep_start = 0.1ns

Baseline configuration

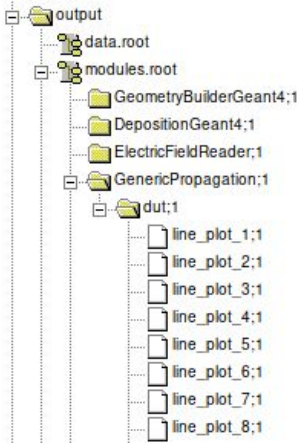
[GenericPropagation]
name = "dut"
temperature = 293K
charge_per_step = 100
integration_time = 100ns



Visualization

Visualization in the framework

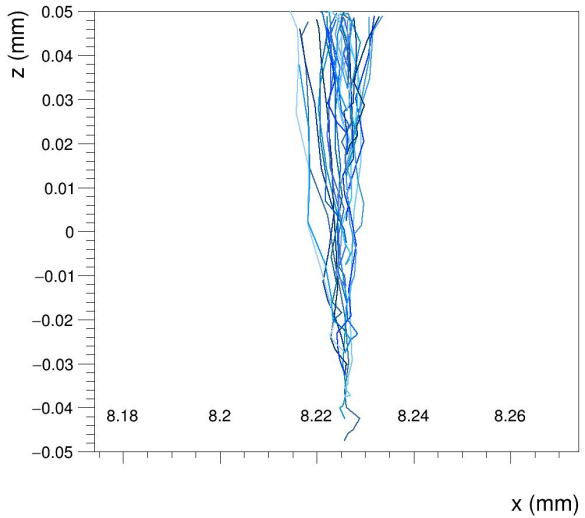
- **output_plots** parameter in many modules
- Produces configurable visualizations, usually at cost of extra computation time
- Plots useful for debugging, gaining an understanding, and for showing results
- By default put in ROOT file *modules.root* file in **output_directory** (sometimes in actual directory for data that cannot be stored in ROOT TTree)
  - Structured per module in top directory
  - Subdirectories indexed based on detector name and input/output name



Trajectory plotting

- 3D plot of trajectory of all propagated carriers (holes and electrons)
- No trajectory plots in ProjectionPropagation (no intermediate steps)
- Slow on events generating many charges and in general for many events!

First examples are simple and do not show full power with custom electric (and magnetic fields) and finer resolution…



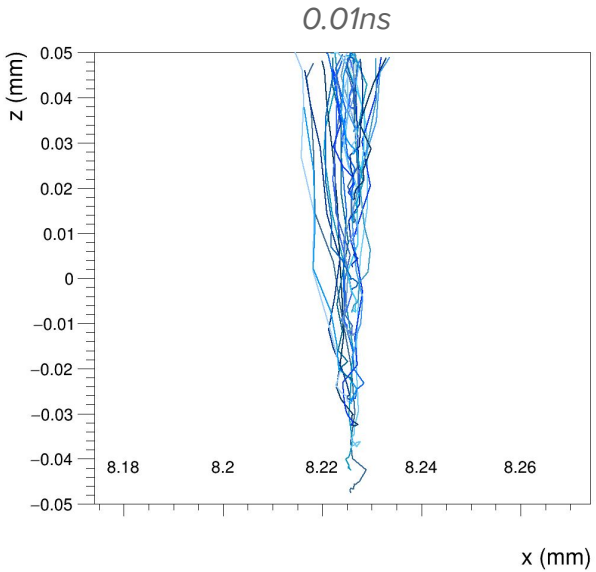

```
[Allpix]
(...)
number_of_events = 1

[DepositionGeant4]
(...)
max_step_length=5um

[GenericPropagation]
name = "dut"
temperature = 293K
charge_per_step = 250
integration_time = 100ns
max_timestep = 0.1ns
output_plots = true
```

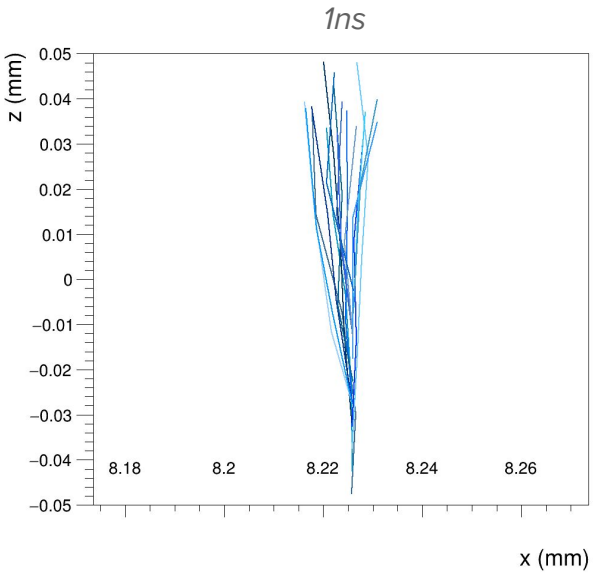
Plot options for generic propagator

Different options

- output_plots_step

*Plot step at least equal to actual time step, thus (recommended not to set below **max_time_step**, the default)*





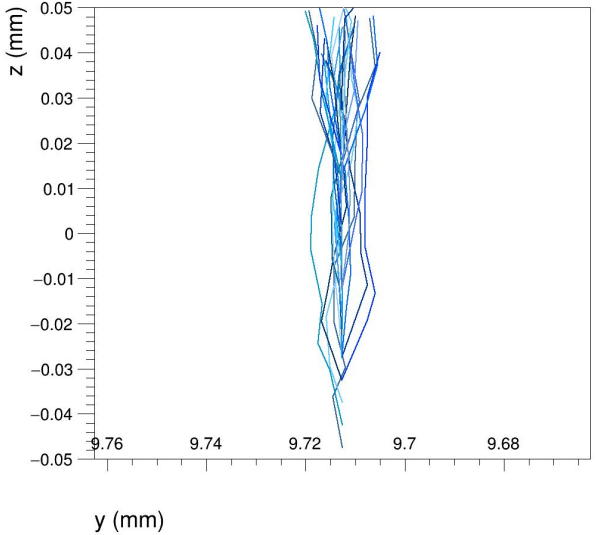
Plot options for generic propagator

Different options

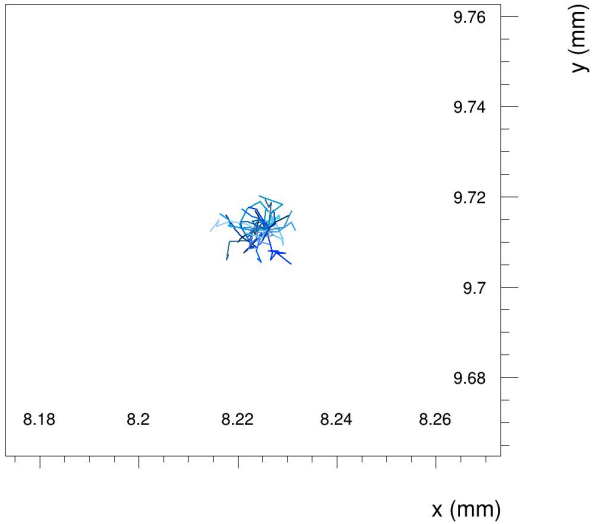
- output_plots_step
- **output_plots_theta**
- **output_plots_phi**

Can also be set later (TCanvas), mostly useful for later animations…

phi = 90deg



theta = 90deg

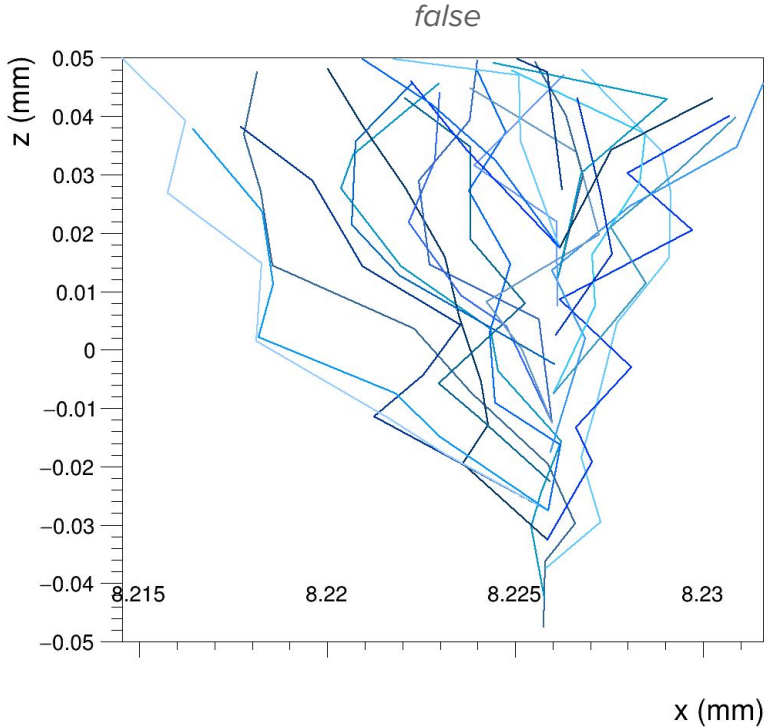


Plot options for generic propagator

Different options

- output_plots_step
- output_plots_theta
- output_plots_phi
- **output_plots_use_equal_scaling**

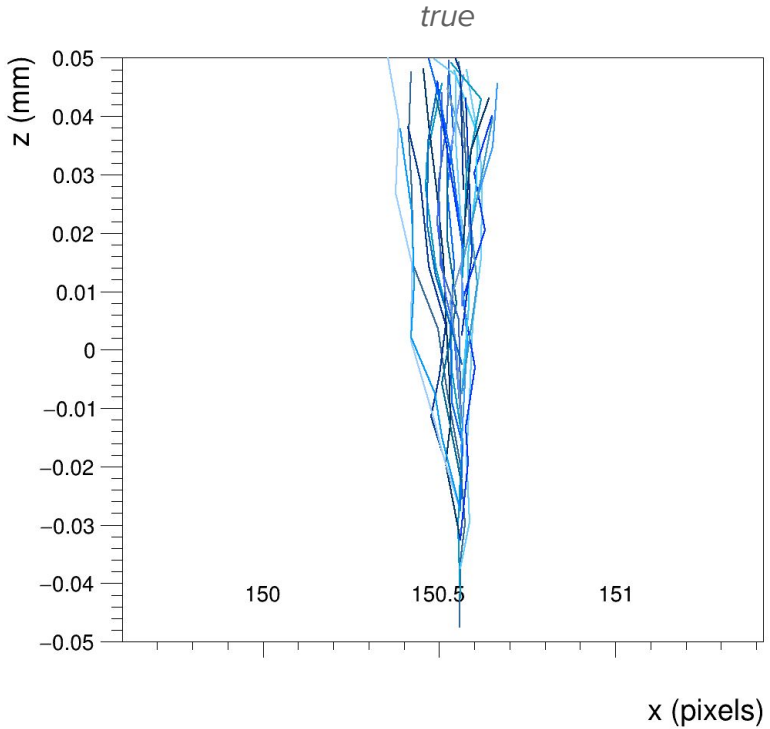
This might give wrong impressions…



Plot options for generic propagator

Different options

- output_plots_step
- output_plots_theta
- output_plots_phi
- output_plots_use_equal_scaling
- **output_plots_use_pixel_units**

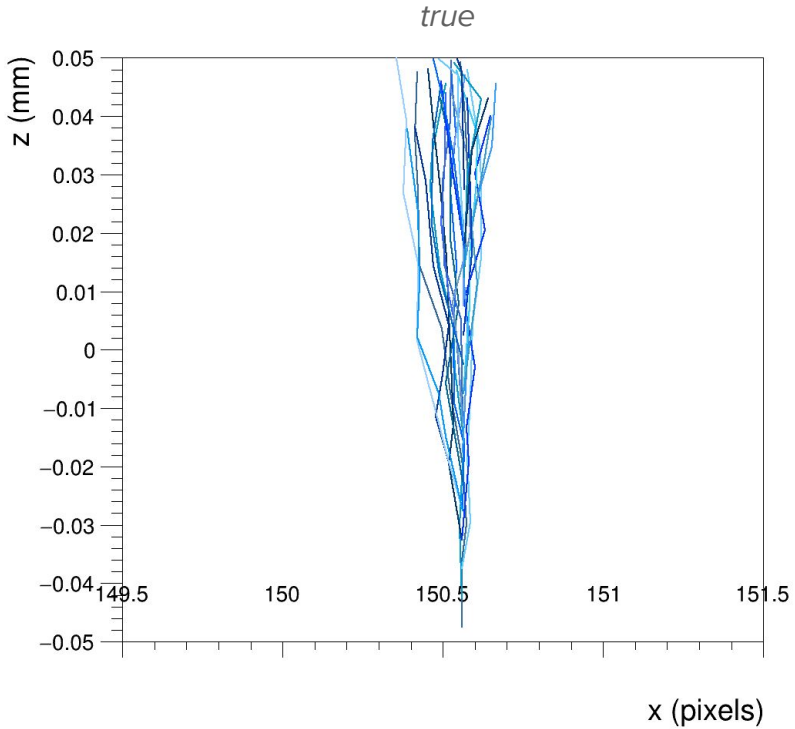


Plot options for generic propagator

Different options

- output_plots_step
- output_plots_theta
- output_plots_phi
- output_plots_use_equal_scaling
- output_plots_use_pixel_units
- **output_plots_align_pixels**

Note the coordinate system: 150 is the center of pixel 150 (starting from zero)!



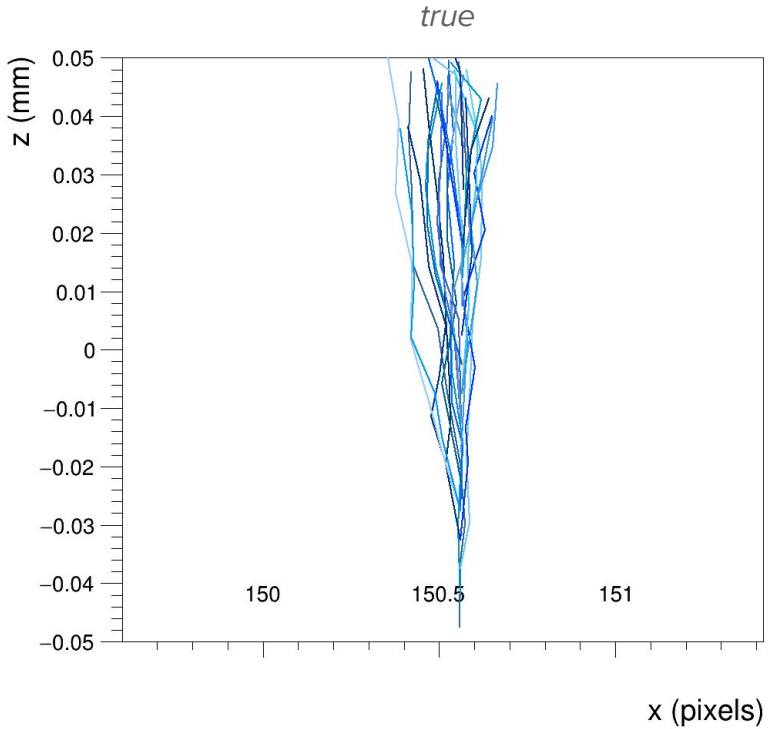
output_plots_use_pixel_units = true

Plot options for generic propagator

Different options

- output_plots_step
- output_plots_theta
- output_plots_phi
- output_plots_use_equal_scaling
- output_plots_use_pixel_units
- output_plots_align_pixels
- **output_plots_lines_at_implants**

All carriers reach implants…



output_plots_use_pixel_units = true

More complete visualization

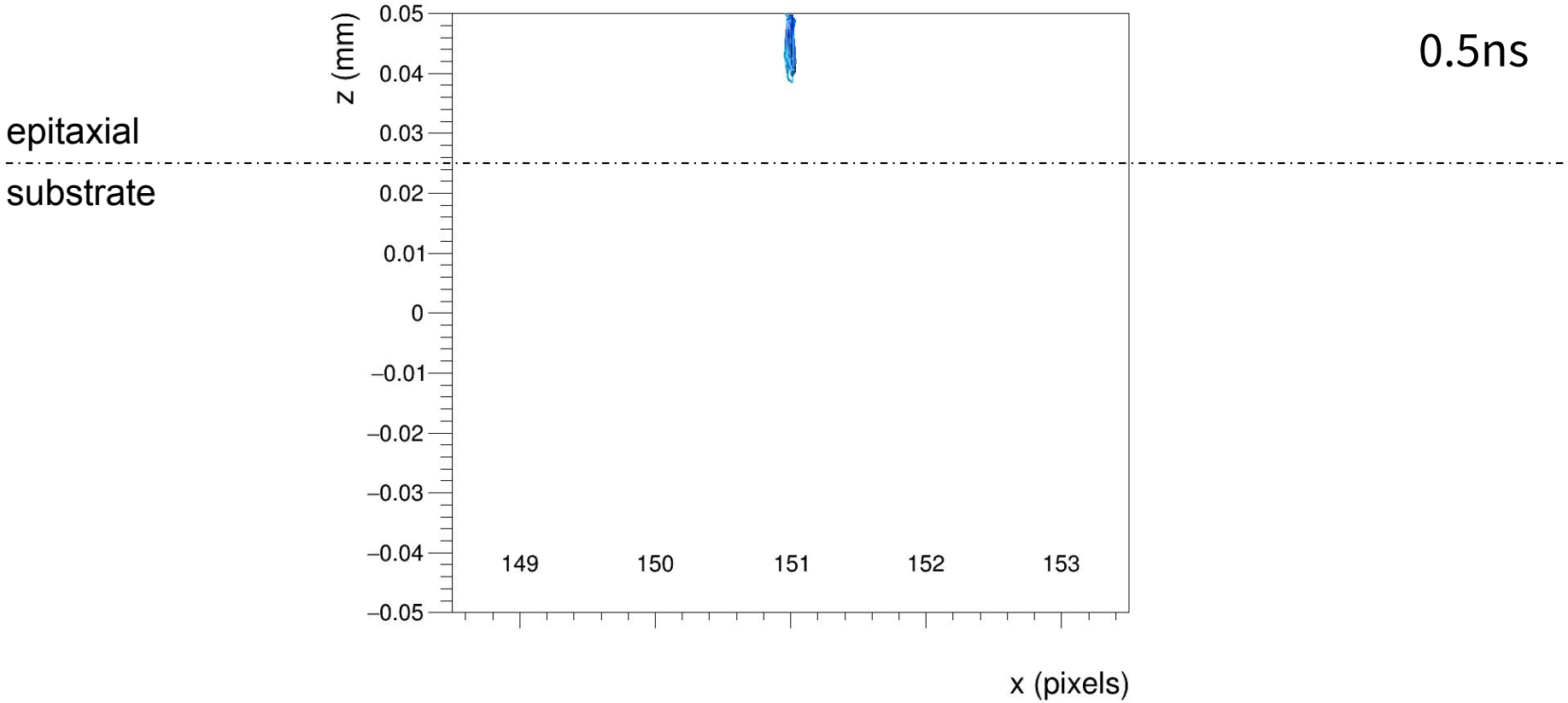
- More interesting visualization† on the following pages:
  - Imported electric field from TCAD
  - One particle, perpendicular incidence in pixel center
  - Projection of drift paths onto X-Z plane (pitch vs. sensor depth)
- Configure to show ‘snapshots’ at different times
  - Set integration time to different moments to change stop time
  - Use **output_plots_lines_at_implants** to only draw charges reaching the sensor implant

Note not ‘real’ snapshots as the random diffusion changes per run

† See also the FAQ for a more beautiful example: https://project-allpix-squared.web.cern.ch/project-allpix-squared/usermanual/allpix-manualch11.html#x12-18300011.6

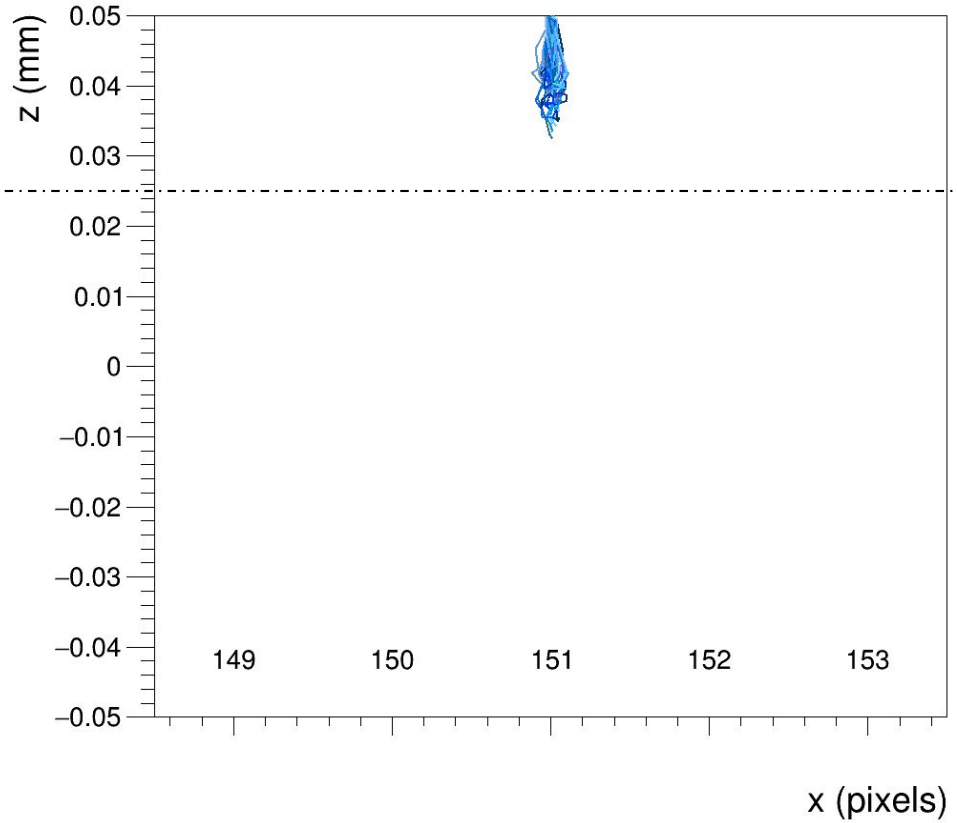
epitaxial

substrate


0.5ns
epitaxial
substrate
z (mm)
0.05
0.04
0.03
0.02
0.01
0
–0.01
–0.02
–0.03
–0.04
–0.05
149
150
151
152
153
x (pixels)

epitaxial

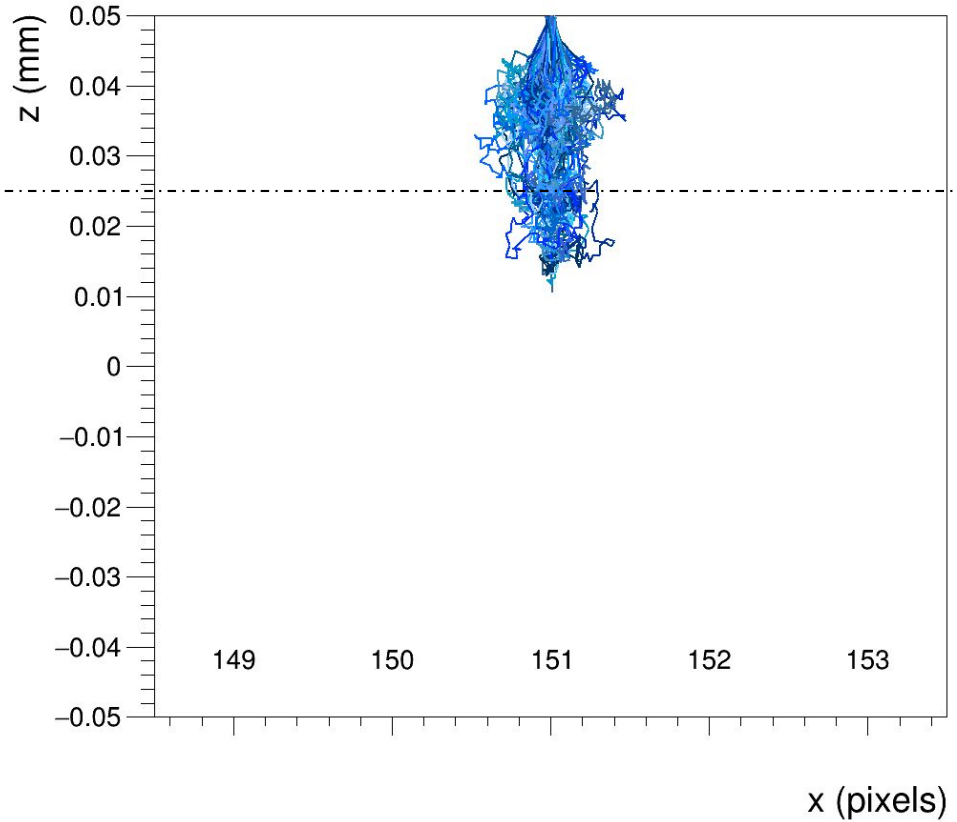
substrate



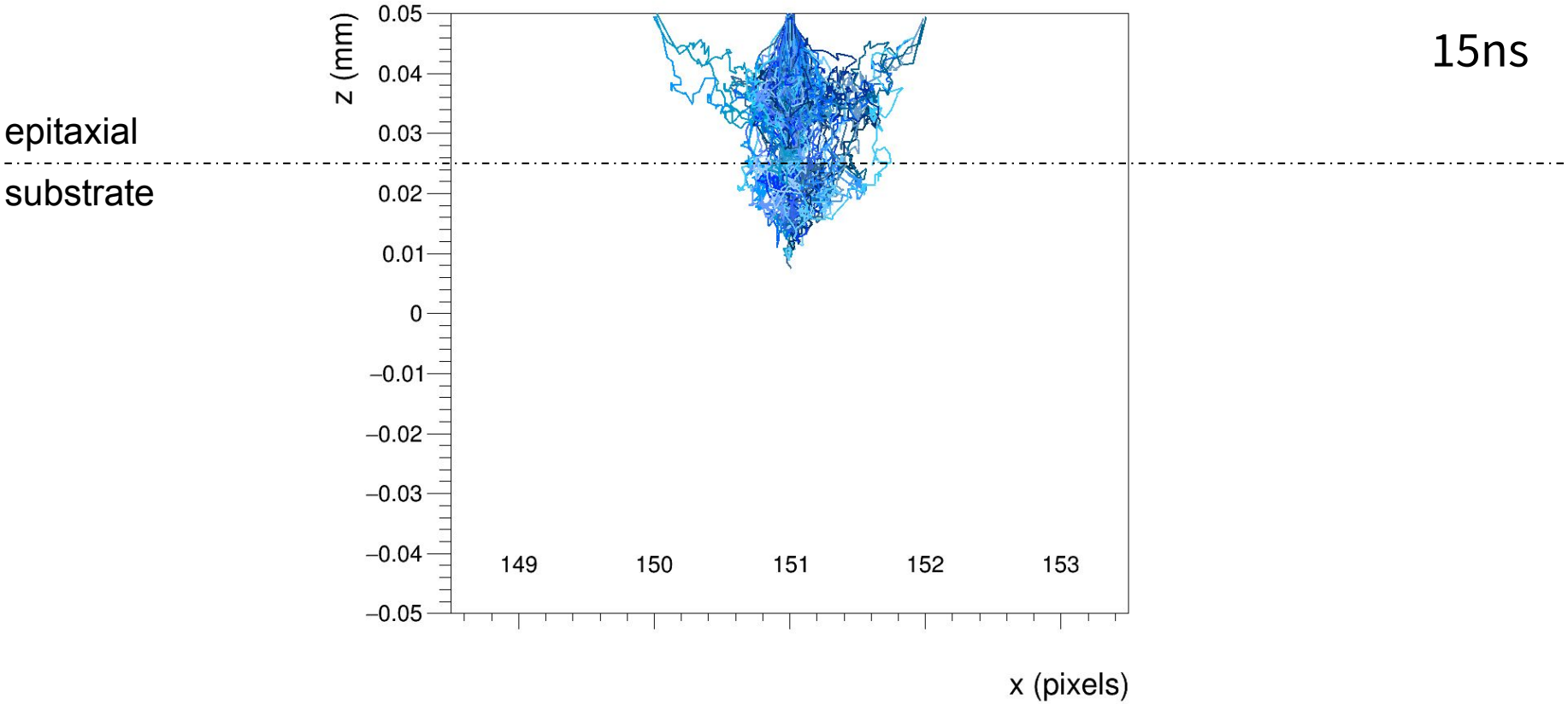
1.5ns

epitaxial

substrate

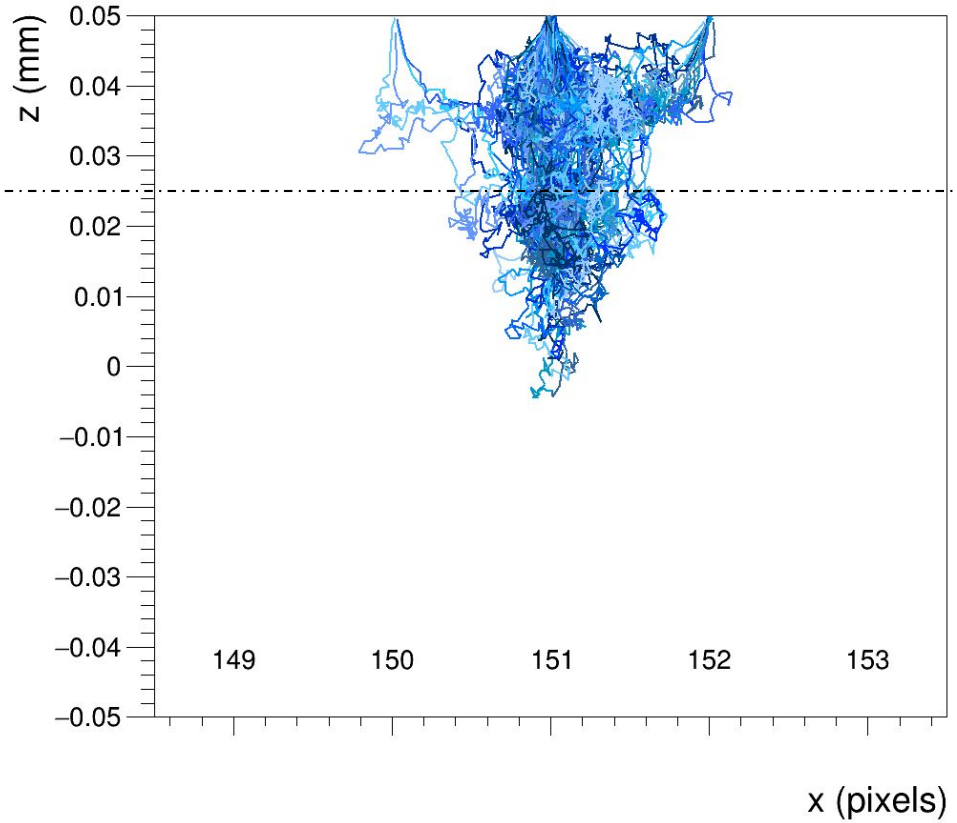


10ns

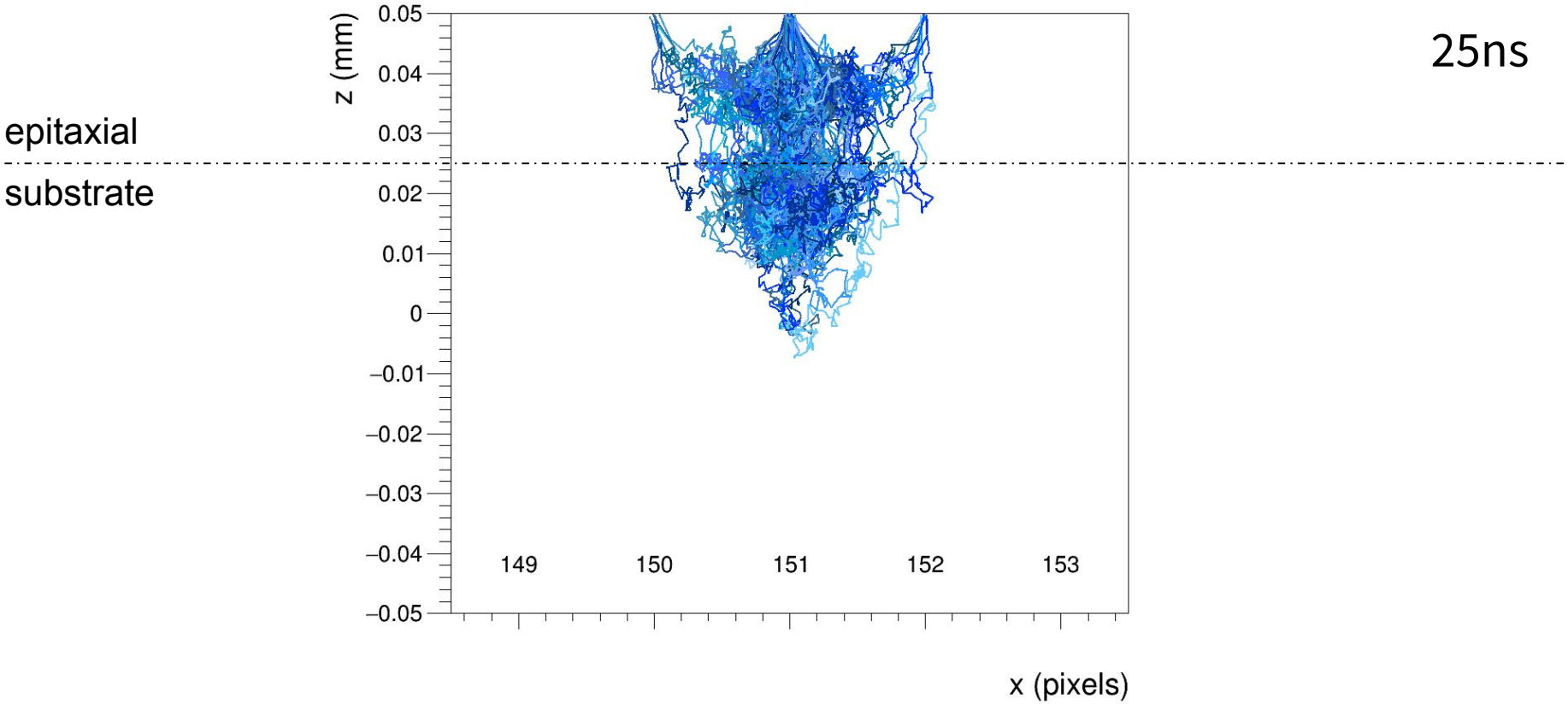

15ns
epitaxial
substrate
z (mm)
0.05
0.04
0.03
0.02
0.01
0
−0.01
−0.02
−0.03
−0.04
−0.05
149
150
151
152
153
x (pixels)

epitaxial

substrate



20ns


25ns
epitaxial
substrate
z (mm)
0.05
0.04
0.03
0.02
0.01
0
−0.01
−0.02
−0.03
−0.04
−0.05
149
150
151
152
153
x (pixels)

Plot animations for generic propagator

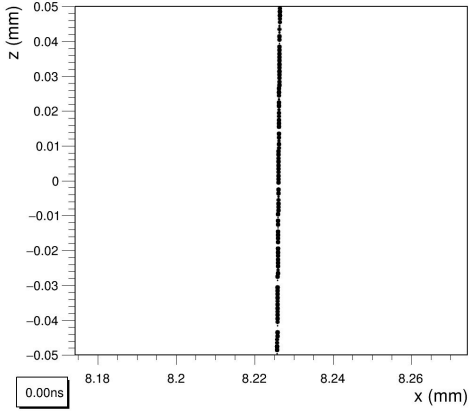
- **output_animations** parameter in GenericPropagation (only with **output_plots**)
- Full animation of carrier propagation over time
  - 3D animation of propagation
  - 2D animation of propagation projected in x, y and z direction
- Slow with single events already, extremely slow with multiple events!

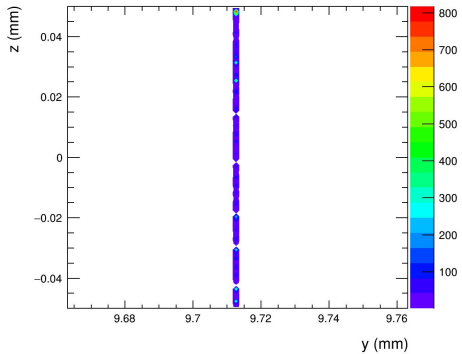
Plot animations for generic propagator

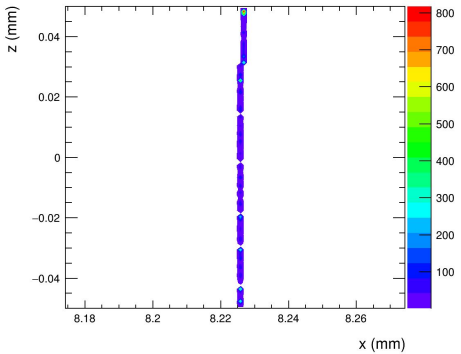
Baseline configuration

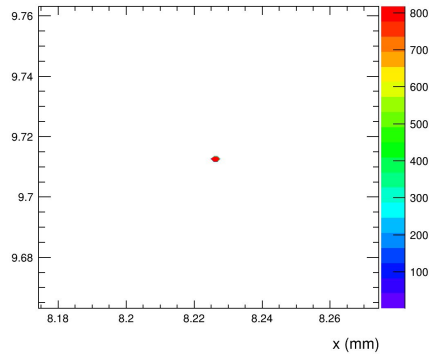
```
[GenericPropagation]
name = "dut"
temperature = 293K
charge_per_step = 50
integration_time = 100ns
output_plots = true
output_animations = true
```

- **output_animations** parameter (with **output_plots**)
- Full animation of carrier propagation over time
  - 3D animation of propagation
  - 2D animation of propagation projected in x, y and z direction
- Slow with single events already, extremely slow with multiple events!









Animation options for generic propagator

3D animation example with different options

[GenericPropagation]

(...)

log_level = “INFO” (set INFO level to show animation progress)

output_plots = true

output_animations = true

timestep_max = 0.1ns

output_plots_step = 0.1ns

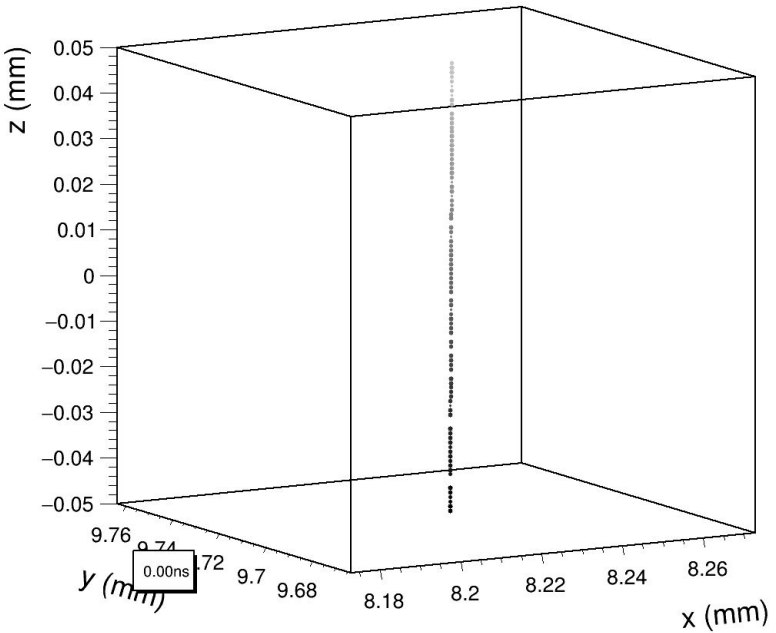
output_plots_theta = 10deg

output_plots_phi = 30deg

output_animations_marker_size = 0.5 (2x smaller than default)

output_animations_time_scaling = 2e9 (2x slower than default)

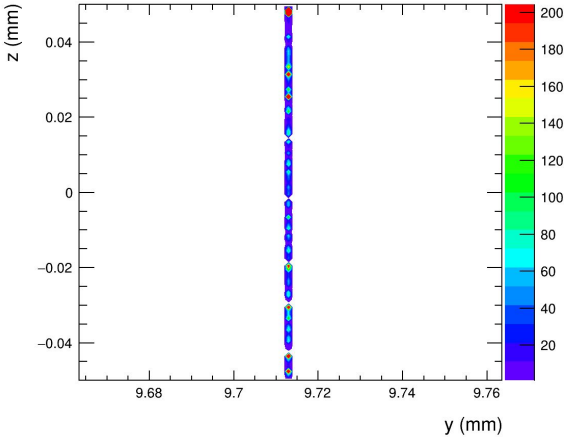
output_animations_color_markers = true

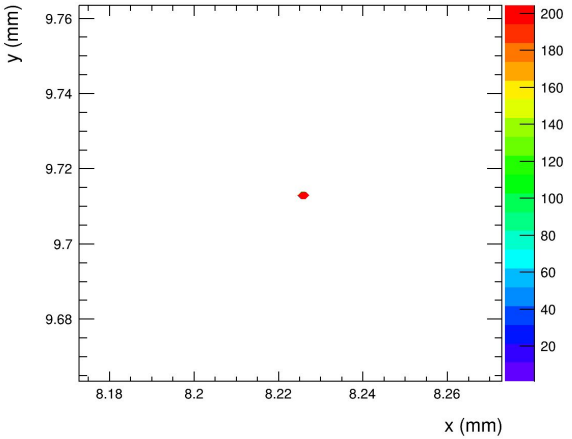


Animation options for generic propagator

2D animation example with different options

[GenericPropagation]
(...)
log_level = “INFO” (set INFO level to show animation progress)
output_plots = true
output_animations = true
timestep_max = 0.1ns
output_plots_step = 0.1ns
output_animations_contour_max_scaling = 40 (4x smaller than default)





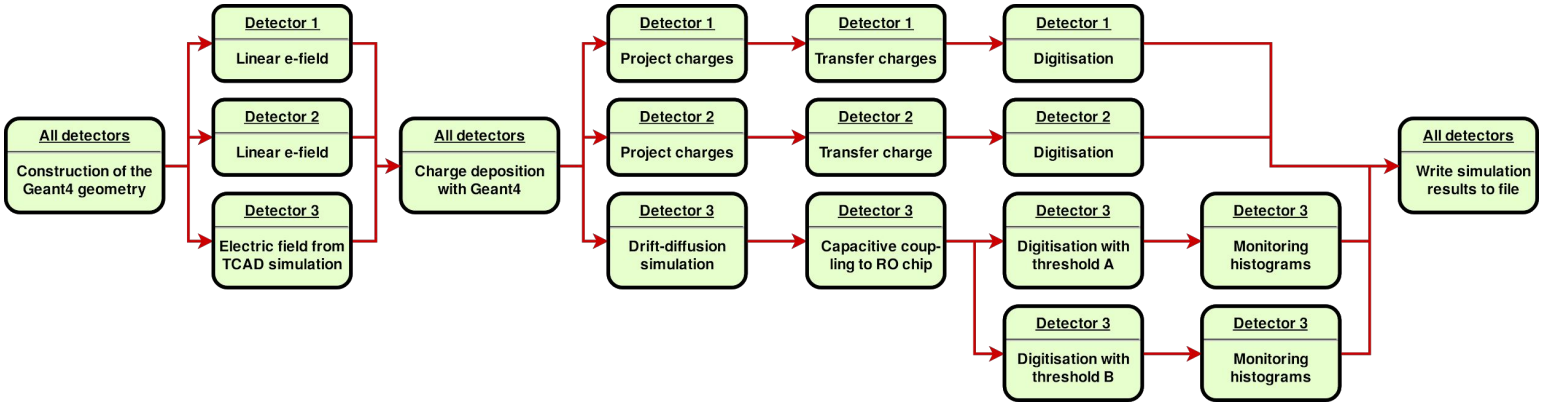
2 minutes to generate for single event!

Simulation Flows

Framework design

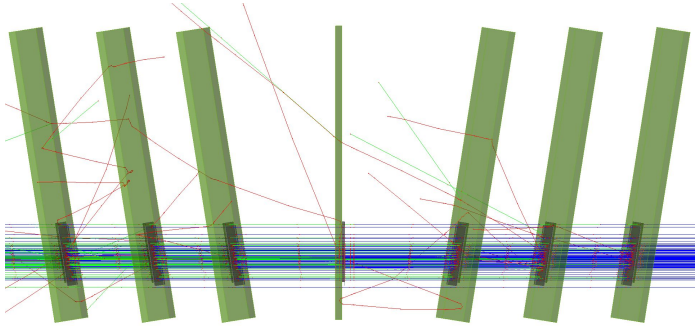
- Modular structure of framework
- Easy-to-use configuration

Flexible options for simulation flows



Multithreaded propagation

- Use thread per instance (detector) for parallel propagation (currently requires telescope)



Guaranteed same results!

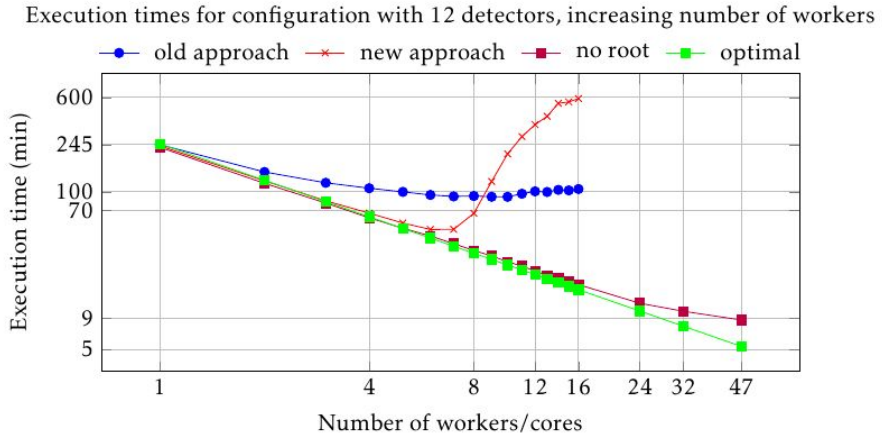
cores	*1*	*8*
time per event	900	408

Only around 2x speedup with 7 detectors ...

```
[Allpix]
log_level = "WARNING"
log_format = "DEFAULT"
detectors_file =
"workshop-geometry-full.conf"
experimental_multithreading = true
number_of_events = 100
random_seed = 0

[GeometryBuilderGeant4]

[DepositionGeant4]
(...)

[ElectricFieldReader]
# name = "dut"
model = "linear"
depletion_voltage = -7V
bias_voltage = -15V

[GenericPropagation]
# name = "dut"
temperature = 293K
charge_per_step = 100
integration_time = 100ns
```

Multithreaded propagation

- Use thread per detector for parallel propagation (currently requires telescope)
- Updates on the way to do multithreading on events†, much more powerful!



† V. Sonesten, https://github.com/allpix-squared/allpix-squared/pull/1

```
[Allpix]
log_level = "WARNING"
log_format = "DEFAULT"
detectors_file =
"workshop-geometry-full.conf"
experimental_multithreading = true
number_of_events = 100
random_seed = 0

[GeometryBuilderGeant4]

[DepositionGeant4]
(...)

[ElectricFieldReader]
# name = "dut"
model = "linear"
depletion_voltage = -7V
bias_voltage = -15V

[GenericPropagation]
# name = "dut"
temperature = 293K
charge_per_step = 100
integration_time = 100ns
```

Different propagators for different detectors

- *dut* detector and *telescope0* until *telescope5*
- Generating instance for type and name

```
[GenericPropagation]
type = "timepix"
temperature = 293K
charge_per_step = 100
integration_time = 100ns

[GenericPropagation]
name = "dut"
temperature = 293K
charge_per_step = 100
integration_time = 100ns
```

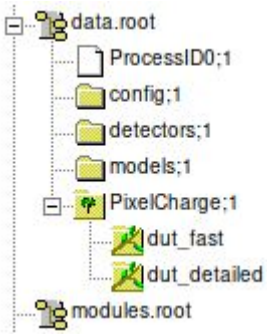
overloading

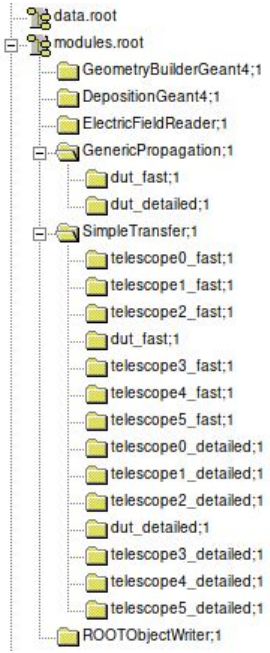
GenericPropagation:telescope0
GenericPropagation:telescope0
(...)
GenericPropagation:dut

```
[ProjectionPropagation]
name = "telescope0", "telescope1",
"telescope2", "telescope3", "telescope4",
"telescope5"
#type = "timepix"
temperature = 293K
charge_per_step = 100
integration_time = 100ns

[GenericPropagation]
name = "dut"
temperature = 293K
charge_per_step = 100
integration_time = 100ns
```

Different propagators for the same detector

- Use input / output mechanism
- Instantiation per input / output
- Object data per output






```
[GenericPropagation]
name = "dut"
output = "fast"
temperature = 293K
charge_per_step = 500
integration_time = 100ns

[GenericPropagation]
name = "dut"
output = "detailed"
temperature = 293K
charge_per_step = 50
integration_time = 100ns

[SimpleTransfer]
input = "fast"
output = "fast"

[SimpleTransfer]
input = "detailed"
output = "detailed"

[ROOTObjectWriter]
include = "PixelCharge"
```

currently bit cumbersome...

Splitting up propagation and later steps

```
[Allpix]
number_of_events = 1000
random_seed = 0
(...)

[GeometryBuilderGeant4]

[DepositionGeant4]
(...)

[ElectricFieldReader]
(...)

[GenericPropagation]
name = "dut"
temperature = 293K
charge_per_step = 500
integration_time = 100ns

[ROOTObjectWriter]
file_name = "step1"
```

1. Compute computation-heavy propagation first
2. Apply custom transfer and digitization in a separate step

|11:25:52.623| (STATUS) [F:ROOTObjectWriter] Wrote 14133149 objects to 16 branches in file:

423MB data for 1000 events

- **number_of_events** and **random_seed** should match
- History links between files†

† [bug] https://gitlab.cern.ch/allpix-squared/allpix-squared/issues/137

```
[Allpix]
(...)
number_of_events = 1000
random_seed = 0

[ROOTObjectReader]
file_name = "temp/output/step1.root"

[SimpleTransfer]
name = "dut"
max_depth_distance = 10um

[DefaultDigitizer]
name = "dut"
electronics_noise = 80e
threshold = 500e
threshold_smearing = 30e
adc_smearing = 350e
adc_resolution = 16
adc_slope = 1.0376e

[ROOTObjectWriter]
file_name = "step2"
include = "PixelHit"
```

Module modifications

Design of a module

- Developer perspective: emphasis on readable and documented code
- Simplicity preferred over too high flexibility and complex optimizations

Design of a module

- Developer perspective: emphasis on readable and documented code
- Simplicity preferred over too high flexibility and complex optimizations
- Walkthrough generic structure of propagator module, no magic…
  - With GenericPropagation as most likely to modify (similar structure in ProjectionPropagation)
  - Usage of configuration component and unit system for easy-to-use parameterization
  - Detector component abstracts the detector configuration
  - Module system together with Messenger logic provide simple, flexible and intuitive integration with other modules and also allow for relatively straightforward multithreading support
  - Use of modern C++ constructs, in particular lambdas, for fast and readable code

Module: *src → modules → GenericPropagation*

Module architecture

- Setup the module
- Constructor
  - *Enabling support for parallelization*
  - Binding deposit messages
  - Initialize random seed
  - Setting default config
  - Store parameters for speed reasons
- Before final instances and fields are loaded
- Use of exceptions

```
GenericPropagationModule::GenericPropagationModule(Configuration& config,
                                                   Messenger* messenger,
                                                   std::shared_ptr<Detector> detector)
    : Module(config, detector), messenger_(messenger), detector_(std::move(detector)) {
    // Enable parallelization of this module if multithreading is enabled
    enable_parallelization();
    // Save detector model
    model_ = detector_->getModel();
    // Require deposits message for single detector
    messenger_->bindSingle(this, &GenericPropagationModule::deposits_message_, MsgFlags::REQUIRED);
    // Seed the random generator with the module seed
    random_generator_.seed(getRandomSeed());
    // Set default value for config variables
    config_.setDefault<double>("spatial_precision", Units::get(0.25, "nm"));
    (...)
    // Set defaults for charge carrier propagation:
    config_.setDefault<bool>("propagate_electrons", true);
    Config_.setDefault<bool>("propagate_holes", false);
    if(!config_.get<bool>("propagate_electrons") && !config_.get<bool>("propagate_holes")) {
        throw InvalidValueError(config_, "propagate_electrons", "No charge carriers selected for propagation, enable
'propagate_electrons' or 'propagate_holes'.");
    }
    config_.setDefault<bool>("ignore_magnetic_field", false);
    // Copy some variables from configuration to avoid lookups:
    temperature_ = config_.get<double>("temperature");
    (...)
    // Parameterization variables from https://doi.org/10.1016/0038-1101(77)90054-5 (section 5.2)
    electron_Vm_ = Units::get(1.53e9 * std::pow(temperature_, -0.87), "cm/s");
    electron_Ec_ = Units::get(1.01 * std::pow(temperature_, 1.55), "V/cm");
    electron_Beta_ = 2.57e-2 * std::pow(temperature_, 0.66);
    (...)
}
```

Module architecture

- Check preconditions and initialize run histograms
- Init
  - Verify electric field setup in sensor
  - Verify magnetic field setup (and possibly ignore) in sensor
  - Initialize a global histogram
- Logging to signal info at different levels

```
void GenericPropagationModule::init() {
    auto detector = getDetector();
    // Check for electric field and output warning for slow propagation if not defined
    if(!detector->hasElectricField()) {
        LOG(WARNING) << "This detector does not have an electric field.";
    }
    // For linear fields we can in addition check if the correct carriers are propagated
    if(detector->getElectricFieldType() == ElectricFieldType::LINEAR) {
        (...)
        // Get the field close to the implants and check its sign:
        auto efield = detector->getElectricField(probe_point); auto direction = std::signbit(efield.z());
        // Compare with propagated carrier type:
        if(direction && !config_.get<bool>("propagate_electrons")) {
            LOG(WARNING) << "Electric field indicates electron collection at implants, but electrons are not propagated!";
        }
        if(!direction && !config_.get<bool>("propagate_holes")) {
            LOG(WARNING) << "Electric field indicates hole collection at implants, but holes are not propagated!";
        }
    }
    // Check for magnetic field
    has_magnetic_field_ = detector->hasMagneticField();
    if(has_magnetic_field_) {
        if(config_.get<bool>("ignore_magnetic_field")) {
            has_magnetic_field_ = false;
            LOG(WARNING) << "A magnetic field is switched on, but is set to be ignored for this module.";
        } else {
            LOG(DEBUG) << "This detector sees a magnetic field.";
            magnetic_field_ = detector_->getMagneticField();
        }
    }
    if(output_plots_step_length_) {
        // Initialize output plot
        step_length_histo_ = new TH1D("step_length_histo", "Step length;length[um];integration steps",100, 0,
static_cast<double>(Units::convert(0.25 * model_->getSensorSize().z(), "um")));
    }
}
```

Module architecture

- Run of a single event
- Logging, statistics and visualization not shown
- Run
  - Get deposits of event and check if propagated
  - Loop over charge steps
  - Propagate set of charges (*next page*)
  - Store result
  - *Update run statistics*
  - Dispatch message
- Multithreading possible

```
void GenericPropagationModule::run(unsigned int event_num) {
  // Create vector of propagated charges to output
  std::vector<PropagatedCharge> propagated_charges;
  // Loop over all deposits for propagation
  for(auto& deposit : deposits_message_->getData()) {
    if((deposit.getType() == CarrierType::ELECTRON && !config_.get<bool>("propagate_electrons")) ||
      (deposit.getType() == CarrierType::HOLE && !config_.get<bool>("propagate_holes"))) continue;
    // Loop over all charges in the deposit
    unsigned int charges_remaining = deposit.getCharge();
    auto charge_per_step = config_.get<unsigned int>("charge_per_step");
    while(charges_remaining > 0) {
      // Define number of charges to be propagated and remove charges of this step from the total
      if(charge_per_step > charges_remaining) charge_per_step = charges_remaining;
      charges_remaining -= charge_per_step;
      // Get position and propagate through sensor
      auto position = deposit.getLocalPosition();
      // Propagate a single charge deposit
      auto prop_pair = propagate(position, deposit.getType());
      position = prop_pair.first;
      // Create a new propagated charge and add it to the list
      auto global_position = detector_->getGlobalPosition(position);
      PropagatedCharge propagated_charge(position, global_position, deposit.getType(), charge_per_step,
deposit.getEventTime() + prop_pair.second, &deposit);
      propagated_charges.push_back(std::move(propagated_charge));
      // Update statistical information (...)
    }
  }
  // Write summary and update statistics (...)
  // Create a new message with propagated charges
  auto propagated_charge_message = std::make_shared<PropagatedChargeMessage>(std::move(propagated_charges),
detector_);
  // Dispatch the message with propagated charges
  messenger_->dispatchMessage(this, propagated_charge_message);
}
```

Module architecture

- Simulate propagation of set of charges
- Simplify overall logic
- Propagate
  - Create lambdas for calculations (physics)
  - Initialize the solver
  - Loop while max time is not reached and deposit remains in sensor
  - Apply integration step
  - Apply diffusion
  - Find final position in sensor

```
std::pair<ROOT::Math::XYZPoint, double> GenericPropagationModule::propagate(
const ROOT::Math::XYZPoint& pos, const CarrierType& type) {
   // Create a runge kutta solver using the electric field as step function
   Eigen::Vector3d position(pos.x(), pos.y(), pos.z());
   // Define a lambda function to compute the carrier mobility (NOTE: often bottleneck)
   auto carrier_mobility = [&](double efield_mag) { return (...); };
   // Define a function to compute the diffusion
   auto carrier_diffusion = [&](double efield_mag, double timestep) -> Eigen::Vector3d { return (...); };
   // Define lambda functions to compute the charge carrier velocity with or without magnetic field
   (...)
   // Create the solver with an RKF5 tableau, using different velocity calculators depending on the magnetic field
   auto runge_kutta = make_runge_kutta(tableau::RK5, (has_magnetic_field_ ? carrier_velocity_withB : carrier_velocity_noB),
timestep_start_, position);
   // Continue propagation until the deposit is outside the sensor
   Eigen::Vector3d last_position = position; double last_time = 0;
   while(detector_->isWithinSensor(static_cast<ROOT::Math::XYZPoint>(position)) && runge_kutta.getTime() <
integration_time_) {
      // Save previous position and time
      last_position = position; last_time = runge_kutta.getTime();
      // Execute a Runge Kutta step
      auto step = runge_kutta.step();
      auto timestep = runge_kutta.getTimeStep(); position = runge_kutta.getValue();
      // Apply diffusion step
      auto efield = detector_->getElectricField(static_cast<ROOT::Math::XYZPoint>(position));
      auto diffusion = carrier_diffusion(std::sqrt(efield.Mag2()), timestep);
      position += diffusion; runge_kutta.setValue(position);
      // Adapt step size to match target precision
      (...)
      runge_kutta.setTimeStep(timestep);
   }
   // Find proper final position in the sensor (...)
   // Return the final position of the propagated charge
   return std::make_pair(static_cast<ROOT::Math::XYZPoint>(position), time);
}
```

Module architecture

- Write run-specific information
- Rather easy for this module :)
- Finalize
  - Write global histogram
  - Logging of statistics with output in preferable units

```
void GenericPropagationModule::finalize() {
   if(output_plots_step_length_) {
      step_length_histo_->Write();
   }

   long double average_time = total_time_ / std::max(1u, total_propagated_charges_);
   LOG(INFO) << "Propagated total of " << total_propagated_charges_ << " charges in " << total_steps_
             << " steps in average time of " << Units::display(average_time, "ns");
}
```

Implementing a new feature

- Carrier lifetime during propagation is currently not simulated in master branch (*more on improvements in later session*)
- Let's implement a simple model! *Disclaimer: this is just an example and very much work-in-progress, implementation very probably not exactly correct…*
- Given the doping concentration, minority carrier lifetime is given by†

$$\tau = \frac{\tau_0}{1 + \frac{N_d}{N_{d0}}}$$

τ_0 = reference lifetime*
N_{d0} = reference doping concentration*

N_d = actual doping concentration

† J.G. Fossum, D.S. Lee, A physical model for the dependence of carrier lifetime on doping density in nondegenerate silicon, Solid-State Electronics, Volume 25, Issue 8, 1982, Pages 741-747
* J. G. Fossum, Computer-aided numerical analysis of silicon solar cells, Solid-State Electronics, Volume 19, Issue 4, 1976, Pages 269-277

Implementing a new feature

- Create storage for constants and create configuration variable *doping_concentration*, use the value zero to indicate the feature is not used

GenericPropagationModule.h

```
(...)

    // Local copies of configuration parameters to avoid costly lookup:
    double temperature_{}, timestep_min_{}, timestep_max_{}, timestep_start_{},
integration_time_{}, doping_concentration_{}, target_spatial_precision_{},
output_plots_step_{};
    bool output_plots_{}, output_plots_step_length_{},
output_plots_lines_at_implants_{};

    // Predefined values for reference charge carrier lifetime and doping
concentration
    double electron_lifetime_reference_;
    double hole_lifetime_reference_;
    double electron_doping_reference_;
    double hole_doping_reference_;

(...)
```

GenericPropagationModule.cpp

```
GenericPropagationModule::GenericPropagationModule(Configuration& config,
                                                   Messenger* messenger,
                                                   std::shared_ptr<Detector> detector)
  : Module(config, detector), messenger_(messenger), detector_(std::move(detector)) {
  (...)

  // Set default value for config variables
  (...)
  config_.setDefault<double>("doping_concentration", 0);
 (...)
  // Copy some variables from configuration to avoid lookups:
  (...)
  doping_concentration_ = config_.get<double>("doping_concentration");
  (...)

  // Reference lifetime and doping concentrations
  electron_lifetime_reference_ = Units::get(1e-5, "s");
  hole_lifetime_reference_ = Units::get(4.0e-4, "s");
  electron_doping_reference_ = Units::get(1e16, "/cm/cm/cm");
  hole_doping_reference_ = Units::get(7.1e15, "/cm/cm/cm");
}
```

Implementing a new feature

- Modify the *propagate* function: create a lambda to determine if carrier is still alive, and break the loop if not alive anymore

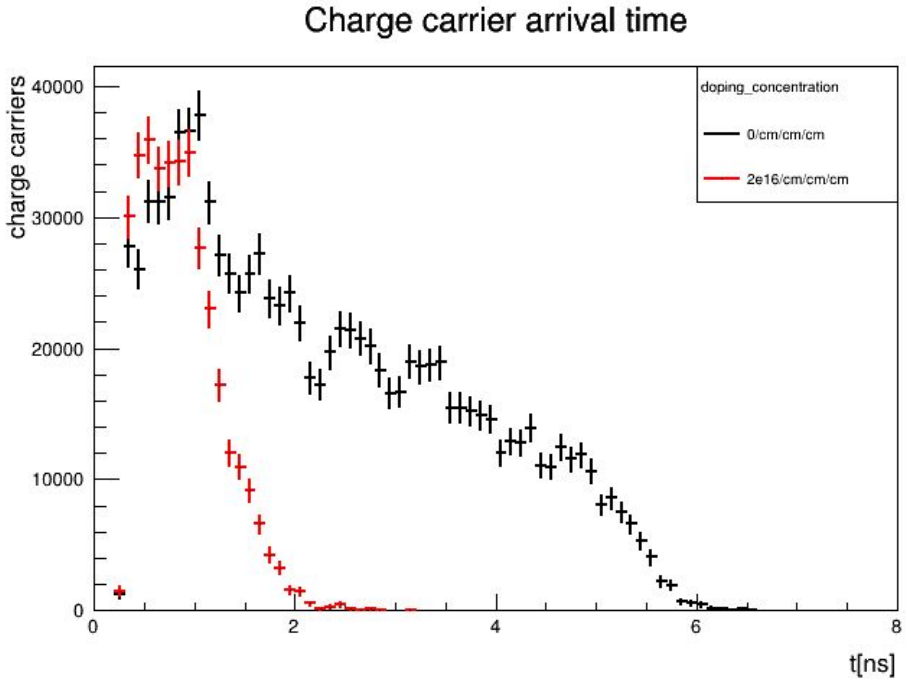
```
auto carrier_alive = [&](double doping_concentration, double time) -> bool {
    // Roll dice for charge carrier survival
    std::uniform_real_distribution<double> survival(0, 1);
    auto lifetime = (type == CarrierType::ELECTRON ? electron_lifetime_reference_ : hole_lifetime_reference_) /
                    (1 + std::fabs(doping_concentration) /
                         (type == CarrierType::ELECTRON ? electron_doping_reference_ : hole_doping_reference_));
    return survival(random_generator_) > std::exp(-(time / lifetime));
};
// Create olver with an RKF5 tableau, using different velocity calculators depending on the magnetic field
(...)
// Continue propagation until the deposit is outside the sensor
(...)
while(detector_->isWithinSensor(static_cast<ROOT::Math::XYZPoint>(position)) &&
      runge_kutta.getTime() < integration_time_) {
    // Check if carrier still alive (if doping concentration is given)
    if(doping_concentration_ > 0 && !carrier_alive(doping_concentration_, runge_kutta.getTime())) {
        break;
    }
    (...)
}
```

Note that the carrier does not actually get removed, its final position will be its last position...

Implementing a new feature

- That's it! Let see the results…
- A more complete, correct and validated implementation of this proposal is currently under development!
- For larger changes to the propagation module, it would generally be easier to copy the whole module, to do comparisons

```
[GenericPropagation]
name = "dut"
temperature = 293K
charge_per_step = 100
integration_time = 100ns
max_timestep = 0.1ns
doping_concentration = 2e16/cm/cm/cm
```



Conclusion

Conclusion

- Charge transport modules in the framework for different scenarios
- Configuration of propagators for different scenarios
- Some useful example simulation flows
- Implementation of a propagation module and possibilities for modifications
- Solid base, but still many useful features to add
    - Induced charge (Ramo)
    - Radiation damage
    - And much more…

Thank you for your attention!