Development of GATE 10 - a new python-based Geant4 application for simulations in medical physics and medical imaging

Nils Krah, on behalf of the GATE collaboration, in particular David Sarrut, Thomas Baudier

Definition: “Monte Carlo Simulation” …

- … in medical physics and medical imaging typically refers to:

“Particle transport simulation”

- Ingredients:

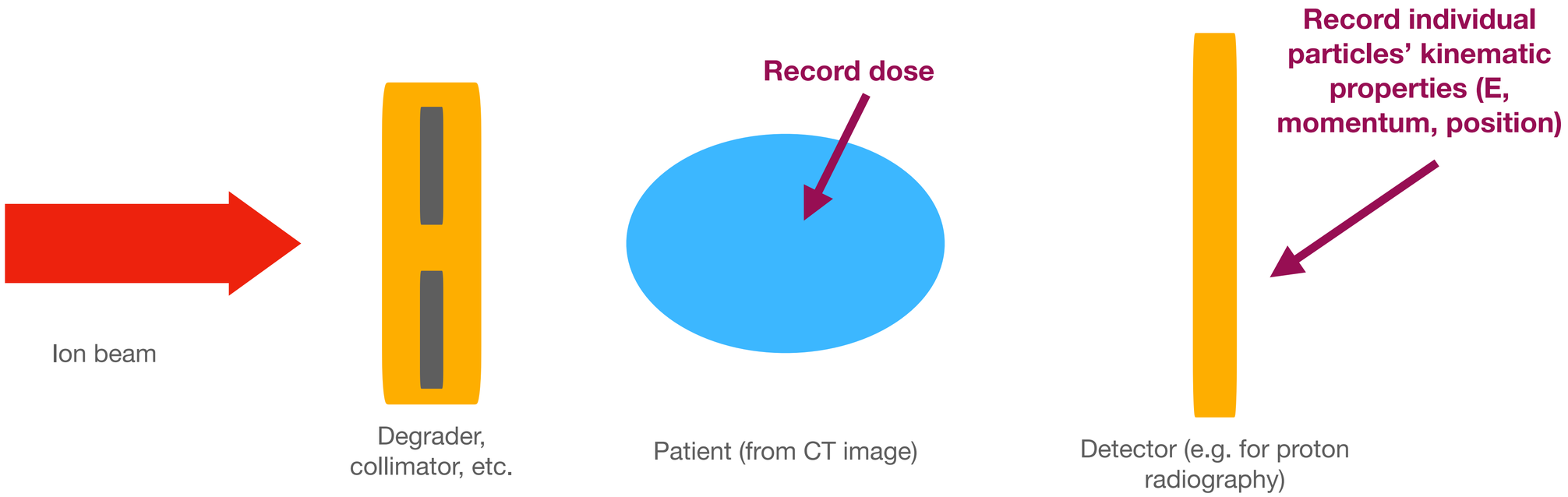
Source (gammas, ions, …)

Geometry (objects, beam line, patient …)

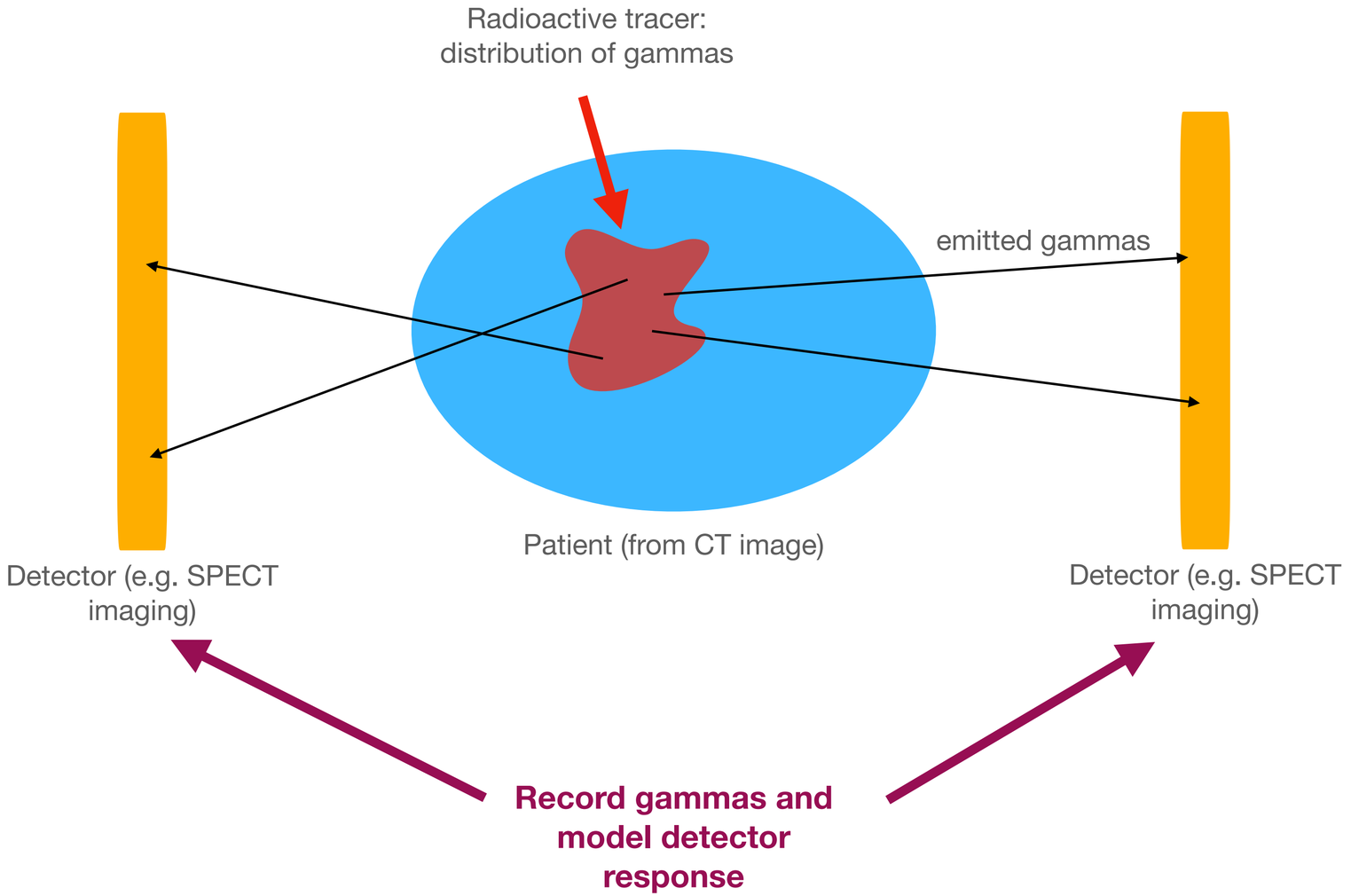
Physics (interactions of particles with the target, nuclear decay)

Output information about physics (dose, particle distribution, detector signal)

Example of “Monte Carlo Simulation” …



Another example of “Monte Carlo Simulation” …



Examples of Monte Carlo codes

Multi-purpose:

- **Geant4, FLUKA, MCNP, …**

Applications built on top of Geant4:

- **TOPAS**
- **GATE version 9.x**
- **new GATE 10**

Application/physics specific:

- **EGS (Electron, photon)**
- **Penelope (electron, photon)**
- **MCsquare (proton therapy)**
- **FRED (mainly fast dose calculation)**
- **…**

What does GATE 10 do?

With GATE 10, the user can build, configure and run a Geant4 simulation from a single python script.

Design goal:

Provide a simple interface to the user while providing access to the rich and long validated Geant4 simulation code.

Let’s take a detour to appreciate this

Ingredients of a Monte Carlo Simulation

Source (gammas, ions, …)

Geometry (objects, beamline, patient …

Physics (interactions of particles with the target, nuclear decay)

Output information about physics (dose, particle distribution, detector signal)

Ingredients: Closer look

Source (gammas, ions, …)

- Position (point source, pencil beam) or spatial distribution (nuclear medicine, nuclear imaging)
- Direction or distribution of directions
- Energy spectrum
- Maybe result of nuclear decay
- Described via parameters or input image (e.g. PET, SPECT)

One primary particle
=
one independent event

Generation of primaries is random process:

Monte Carlo

Ingredients: Closer look

Geometry

- Placement and orientation of objects
- Shape of objects
- Material properties
- Object-specific physics
- Geometrically constructed (box, sphere)
- or derived from 3D voxelized image

Hierarchy of objects: one inside another

Ingredients: Closer look

Physics

- Set of possible processes per particle
- Each process has a probability (cross section)
- Depends on material, energy, etc.
- Process might generate secondary particles
- Geant4 provides physics models: data-based, tabulated, calculated etc.
- The user might partially disregard physics on purpose (e.g. production cuts)

Step-wise particle transport

Per step:

Select process and generate its outcome

Ingredients: Closer look

Output information about physics → **“Actors” in GATE**

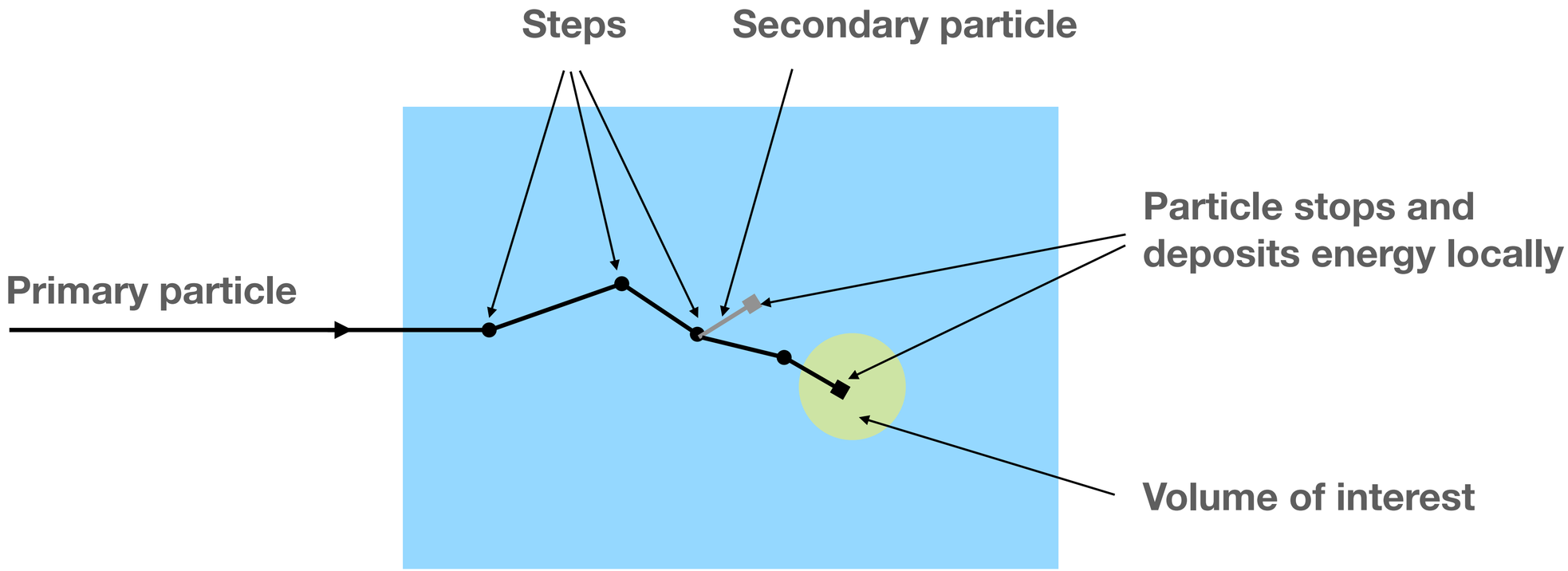
Examples:

- Accumulate dose deposited inside a small cylinder inside a water box
- Record position and direction of all particles crossing a plane
- Record light output of a scintillator
  … and apply post-processing chain
- Record all prompt gammas generated by a proton beam

Mechanism:

Hook into the step-wise particle transport

Example: dose deposit in a volume

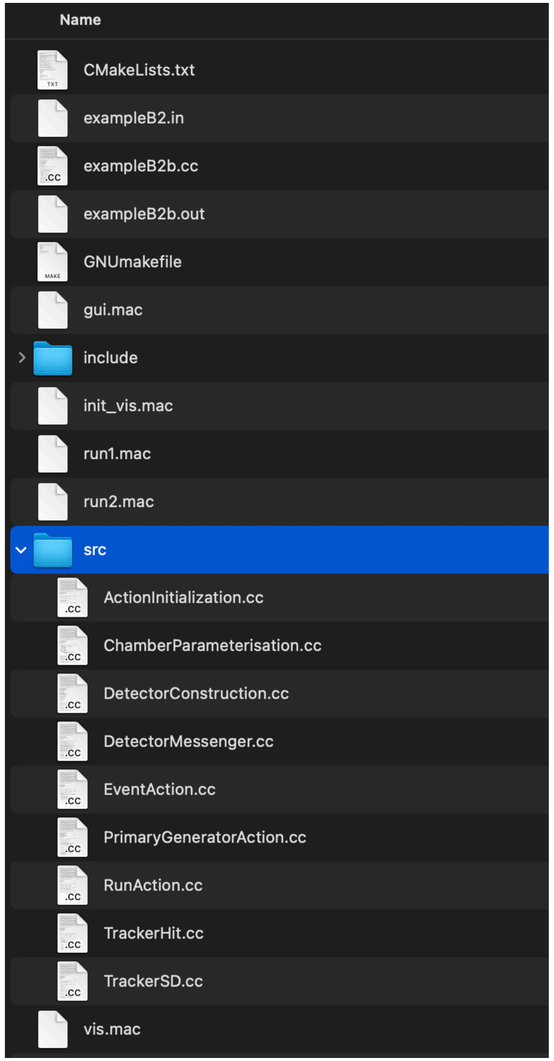


GATE Actor: At every step, get the position, check if inside the volume of interest, get the deposited energy, and add to the accumulated energy.

Side remark:
The main effort of a Monte Carlo simulation is not “Monte Carlo”, but consistently keeping track of particles, processes, geometry.

How would Geant4 work?

Basic G4 example B2b



Write multiple C++ files for geometry, source, physics, output recording

Compile into a program

Run simulation

Process output

Complicated for users in our field

How does GATE 10 work?

```
sim = gate.Simulation()

cm = gate.g4_units("cm")
mm = gate.g4_units("mm")
MeV = gate.g4_units("MeV")

waterbox = sim.create_and_add_volume("Box", "Waterbox")
waterbox.size = [40 * cm, 40 * cm, 40 * cm]
waterbox.translation = [0 * cm, 0 * cm, 25 * cm]
waterbox.material = "G4_WATER"

source = sim.add_source("GenericSource", "Default")
source.particle = "proton"
source.energy.mono = 150 * MeV
source.position.radius = 10 * mm
source.direction.type = "momentum"
source.direction.momentum = [0, 0, 1]
source.n = 20000

sim.add_actor("SimulationStatisticsActor", "Stats")

sim.run()

stats = sim.output.get_actor("Stats")
print(stats)
```

Write a few lines in python for geometry, source, physics, output recording

Execute the python script

Done

Much easier for users in our field

Can be run in interactive python terminal, e.g. jupyter notebook

How do I install GATE 10?

In a terminal, type:

pip install opengate

The installation …

- … takes care of architectures (linux, osx, win*) and python version
- … installs dependences: Geant4, ITK, QT, uproot, etc.
- … downloads Geant4 data and test data

*coming soon

GATE 10 under the hood

Folder g4_bindings	Geant4 binding from C++ to Python (expose functions, classes) ; pybind11
Folder opengate_lib	Core classes (running): source, scorers etc

Folder opengate	User UI (initialisation)

Courtesy of David Sarrut

GATE 10 under the hood

Setup simulation: Volumes, sources, actors, physics etc.

GATE 10 starts “engines”
and creates Geant4 objects via the library interface

Geant4 executes the simulation via the G4RunManager

GATE 10 releases all G4 objects, destroys the
G4RunManager, and closes the engines

Output is available on python side via the Simulation object

Short focus on “Digitizers”

Example processing chain for a scintillator crystal:

Record individual photons in the crystal: Hits

Sum up individual photons

Ideal light signal in the crystal

Apply energy blurring

Apply temporal blurring

Measured light signal in the crystal

Short focus on “Digitizers”: GATE 10 code

Example processing chain for a scintillator crystal:

```
hc = sim.add_actor("DigitizerHitsCollectionActor", "Hits")
hc.mother = [ crystal1.name, crystal2.name ]
hc.output = ”output.root"
hc.attributes = [
        "PostPosition",
        "TotalEnergyDeposit",
        "PreStepUniqueVolumeID",
        "GlobalTime"]
```

Sum up individual photons

```
sc =
sim.add_actor("DigitizerAdderActor",
"Singles")
sc.input_digi_collection = "Hits"
sc.policy =
"EnergyWeightedCentroidPosition"
sc.output = hc.output
```

Apply energy blurring

```
bc1 = sim.add_actor("DigitizerBlurringActor",
"Singles_1")
bc1.output = ""
bc1.input_digi_collection = "Singles"
bc1.blur_attribute = "GlobalTime"
bc1.blur_method = "Gaussian"
bc1.blur_fwhm = 100 * ns
```

Apply temporal blurring

```
bc2 =
sim.add_actor("DigitizerBlurringActor",
"Singles_2")
bc2.output = hc.output
bc2.input_digi_collection = bc1.name
bc2.blur_attribute =
"TotalEnergyDeposit"
bc2.blur_method = "InverseSquare"
bc2.blur_resolution = 0.18
bc2.blur_reference_value = 511 * keV
```

How is GATE 10 different from GATE 9 or TOPAS?

GATE 10 uses python scripts
while GATE 9 and TOPAS read in static input files.

Advantage:

- Numerical operation are easy, e.g. repeat volumes, rotate or shift detector, beam weights
- Easy to interface with other software, e.g. TPS
- Implement complex systems, e.g. PET scanner, as python module and share it
- Numerous enhancements (speed, IA integration, digitizers, etc)

Example of how to use a complex system

From GATE 10 community contributions:
pet_siemens_biograph.py

```
import opengate.contrib.pet_siemens_biograph as pet_biograph

sim = gate.Simulation()

# This loads a predefined PET scanner
pet = pet_biograph.add_pet(sim, “pet")

# This loads a predefined digitizer chain
pet_biograph.add_digitizer(
    sim, pet.name,"test_pet.root",
    singles_name="Singles"
      )

…
```

New features in GATE 10 compared to GATE 9.x

- Boolean volumes
- TAC source (time activated curves)
- beta+ sources
- Acceptance Angle
- Angular Response Functiions with Neural Networks
- GAN sources (conditional, spect, pet, voxelized)
- Gamma ion source (soon)
- Some (almost) ready-to-use models :
  - SPECT: GE NM 670, Siemens Intevo Bold, Spectrum Veriton (soon)
  - PET: Vereos Philips, Vision Siemens (soon)

Contribute to GATE 10

GATE 10 is an open-source community project, just as GATE 9.x has been. Any contribution is welcome!

https://github.com/OpenGATE/opengate

More than 100 tests/examples in the repository to get you started.

Documentation (in progress). You can edit the doc online.

https://opengate-python.readthedocs.io/

Contribute to GATE 10 as a user

- Install GATE 10 on your machine and starts working with it.
- pip install --pre opengate
- Ask questions via the GATE mailing list: see http://www.opengatecollaboration.org
- Report issues via github: https://github.com/OpenGATE/opengate/issues

Every feedback is welcome!

Contribute to GATE 10 as a developer

- Fork the opengate repo into your own github repo
- Create a branche for the new contribution you are working on
- When done, create a pull request.
- Check the developers doc for details:

https://opengate-python.readthedocs.io/en/latest/developer_guide.html#installation-for-developers

Example contribution from MedAustron

- The medical physics team at MedAustron is implementing a new version of their ion dose calculation pipeline based on GATE 10:

IDEAL v2

IDEAL v1

- Simulations: **GateRTion v1** (C++/mac)
- **3** different python **programs** to:
  - Preprocess the DICOM input
  - Write mac file to start the simulations in Gate
  - Postprocess dose output
- **Single threaded**
  - RAM limitation
  - need for external parallelization tools (HT Condor)

IDEAL v2

- Simulations: **Gate10** (python interface, C++ core)
- **Single** python **program**:
  - Can directly read DICOM files
  - Simulations are started directly from python
  - Dose output available in python
- **Multi-threading** possible
  - Potentially no need for external parallelization tools

Courtesy of Martina Favaretto

Example contribution from MedAustron

- The medical physics team at MedAustron is implementing a new version of their ion dose calculation pipeline based on GATE 10:

Toward IDEAL v2: treatment plan source

- Gate10 Treatment Plan source: array of Pencil Beam sources, one for each spot
  - Only on python side, no Cpp implementation needed

- Initialization:
  - Spots to scan
    - from DICOM RT plan file path
    - from .txt → backward compatibility Gate 9
    - each spot manually → testing and debugging
  - Beamline model
    - set Pencil Beam energy-dependent parameters
  - Total number of particles to simulate

ADD TO SIMULTION

From RT dicom path

```
tps = gate.TreatmentPlanSource("RT_plan", sim)
tps.set_beamline_model(IR2HBL)
tps.set_particles_to_simulate(nSim)
tps.set_spots_from_rtplan(rt_plan)
tps.initialize_tpsource()
```

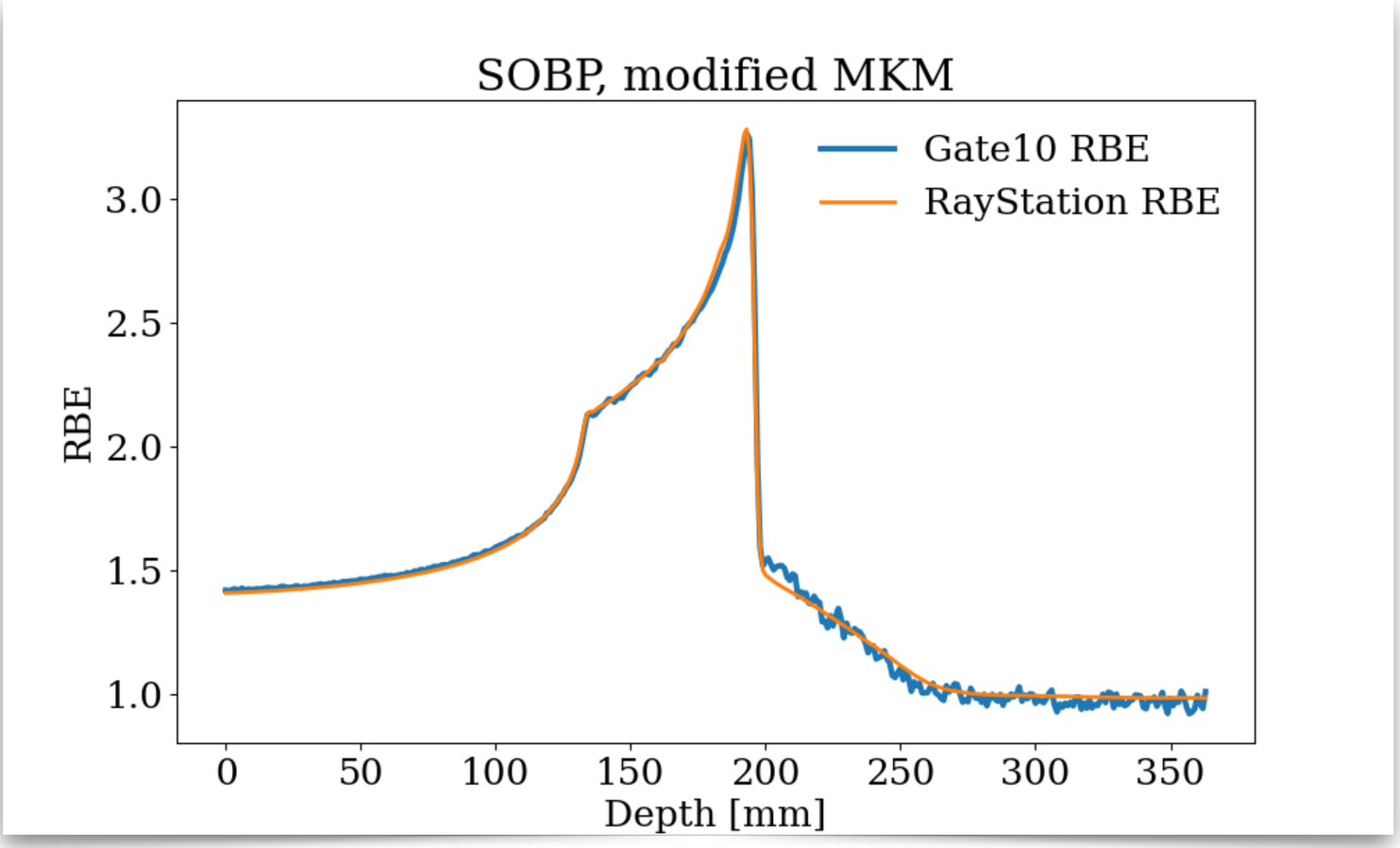
From .txt

```
spots, ntot, energies, G = gate.spots_info_from_txt(
    ref_path / "TreatmentPlan4Gate.txt", "ion 6 12"
)
tps.set_spots(spots)
```


Courtesy of Martina Favaretto

Another contribution from MedAustron

- The medical physics team at MedAustron is implementing radiobiological models, e.g. the mMKM model:



Courtesy of Yihan Jia

Contribute to GATE 10: Hackathons

Participate in a hackathon event, like the last one in Krakow in April 2023.

https://shorturl.at/bfoY5

Example enhancement from that hackathon:

Efficiency actor implemented by Aurélien Coussat, thanks!

Summary

- GATE 10 has evolved from the previous developments of the OpenGATE collaboration.
- It offers the user a simple python interface to complex Geant4 functionality.
- Installation is extremely simple.
- The project is under lively development.
- Contributions on all levels are welcome.

Thank you

on behalf of the GATE collaboration