GATE status and future perspectives

David Sarrut, on behalf of the collaboration

Current release is 9.2 – April 2022

- Mostly consolidation of 9.1
- With Geant4 11
  - Gate is only evaluated with this Geant4 version
  - May work with previous, but no guarantee (and not maintained)
- More than 70 contributors (since 2012 only)
- Automated compilation check on Linux + OSX
- 20+ benchmarks
- Docker + VirtualMachine

https://github.com/OpenGATE/Gate

Future developments

Introducing …

Python GATE* !

NOT the definitive name …

Goal

A (future) new GATE version

- Macro files are replaced by Python scripts
- Geant4 multithreading
- Simplified installation (one line)
- Numerous enhancements (speed, IA integration, digitizer, etc)

Still … we keep

- Open-source, open-mind
- OpenGate community
- Collaborative work
- Research oriented in Medical Physics (imaging, dosimetry)

Current GATE serie 9.2 (c++) will be **continued**

Future GATE serie 10.0 (python/c++) is an additional **experimental** project

Transition period: two versions.

Interests

- Easy, one line installation
- Writing simulation with Python is **much more** efficient than with macros
  - Sharing simplified: users-developed modules (e.g. IEC phantoms, detectors, linac etc.)
  - Easy access to whole Python ecosystem (AI !)
- Speed:
  - G4 engine: still same time, no additional time penalty
  - G4 multithread
  - Current GATE overhead: hopefully reduced
- Faster development time, better separation:
  - Python: user parameters, options, I/O etc
  - C++: core processing only
- Advanced features:
  - Dynamic trigger: call user function during runtime (time penalty)

```python
# create the simulation
sim = gam.Simulation()

# main options
ui = sim.user_info
ui.g4_verbose = False
ui.g4_verbose_level = 1
ui.visu = False

# add a material database
sim.add_material_database(paths.data / 'GateMaterials.db')

# units
m = gam.g4_units('m')
cm = gam.g4_units('cm')
MeV = gam.g4_units('MeV')
Bq = gam.g4_units('Bq')
mm = gam.g4_units('mm')

#  change world size
world = sim.world
world.size = [1 * m, 1 * m, 1 * m]

# add a simple fake volume to test hierarchy
# translation and rotation like in the Gate macro
fake = sim.add_volume('Box', 'fake')
fake.size = [40 * cm, 40 * cm, 40 * cm]
fake.material = 'G4_AIR'
fake.color = [1, 0, 1, 1]
fake.rotation = Rotation.from_euler('x', 20, degrees=True).as_matrix()

# image
patient = sim.add_volume('Image', 'patient')
patient.image = paths.data / 'patient-4mm.mhd'
patient.mother = 'fake'
patient.material = 'G4_AIR'  # material used by default
patient.voxel_materials = [[-2000, -900, 'G4_AIR'],
                           [-900, -100, 'Lung'],
                           [-100, 0, 'G4_ADIPOSE_TISSUE_ICRP'],
                           [0, 300, 'G4_TISSUE_SOFT_ICRP'],
                           [300, 800, 'G4_B-100_BONE'],
                           [800, 6000, 'G4_BONE_COMPACT_ICRU']]
# or alternatively, from a file (like in Gate)
vm = gam.read_voxel_materials(paths.gate_data / 'patient-HU2mat-v1.txt')
vm[0][0] = -2000
assert vm == patient.voxel_materials
patient.voxel_materials = vm
# write the image of labels (None by default)
patient.dump_label_image = paths.output / 'test009_label.mhd'

# default source for tests
source = sim.add_source('Generic', 'mysource')
source.energy.mono = 130 * MeV
source.particle = 'proton'
source.position.type = 'sphere'
source.position.radius = 10 * mm
source.position.translation = [0, 0, -14 * cm]
source.activity = 10000 * Bq
source.direction.type = 'momentum'
source.direction.momentum = [0, 0, 1]

# cuts
c = sim.get_physics_user_info().production_cuts
c.patient.electron = 3 * mm

# add dose actor
dose = sim.add_actor('DoseActor', 'dose')
dose.save = paths.output / 'test009-edep.mhd'
dose.mother = 'patient'
dose.size = [99, 99, 99]
dose.spacing = [2 * mm, 2 * mm, 2 * mm]
dose.img_coord_system = True
dose.translation = [2 * mm, 3 * mm, -2 * mm]
dose.hit_type = 'random'

# add stat actor
stats = sim.add_actor('SimulationStatisticsActor', 'Stats')
stats.track_types_flag = True

# create G4 objects
sim.initialize()

# print info
print(sim.dump_volumes())

# verbose
sim.apply_g4_command('/tracking/verbose 0')

# start simulation

sim.start()
```

```python
import gam_gate as gam
import gam_g4 as g4
import pathlib
import os
```

```python
# add a material database
sim.add_material_database(paths.data / 'GateMaterials.db')

# units
m = gam.g4_units('m')
cm = gam.g4_units('cm')
MeV = gam.g4_units('MeV')
Bq = gam.g4_units('Bq')
mm = gam.g4_units('mm')

# change world size
world = sim.world
world.size = [1 * m, 1 * m, 1 * m]
```

```python
# add a simple fake volume to test hierarchy
# translation and rotation like in the Gate macro
fake = sim.add_volume('Box', 'fake')
fake.size = [40 * cm, 40 * cm, 40 * cm]
fake.material = 'G4_AIR'
fake.color = [1, 0, 1, 1]
fake.rotation = Rotation.from_euler('x', 20, degrees=True).as_matrix()

# image
patient = sim.add_volume('Image', 'patient')
patient.image = paths.data / 'patient-4mm.mhd'
patient.mother = 'fake'
patient.material = 'G4_AIR'  # material used by default
patient.voxel_materials = [[-2000, -900, 'G4_AIR'],
                           [-900, -100, 'Lung'],
                           [-100, 0, 'G4_ADIPOSE_TISSUE_ICRP'],
                           [0, 300, 'G4_TISSUE_SOFT_ICRP'],
                           [300, 800, 'G4_B-100_BONE'],
                           [800, 6000, 'G4_BONE_COMPACT_ICRU']]
# or alternatively, from a file (like in Gate)
vm = gam.read_voxel_materials(paths.gate_data / 'patient-HU2mat-v1.txt')
vm[0][0] = -2000
assert vm == patient.voxel_materials
patient.voxel_materials = vm
# write the image of labels (None by default)
patient.dump_label_image = paths.output / 'test009_label.mhd'
```

```python
# default source for tests
```

```python
# create the simulation
sim = gam.Simulation()

# main options
ui = sim.user_info
ui.g4_verbose = False
ui.g4_verbose_level = 1
ui.visu = False
ui.random_engine = 'MersenneTwister'
ui.random_seed = 'auto'
ui.number_of_threads = 5  paths.data / 'GateMaterials.db')

# units
m = gam.g4_units('m')
cm = gam.g4_units('cm')
MeV = gam.g4_units('MeV')
Bq = gam.g4_units('Bq')
mm = gam.g4_units('mm')

# change world size
world = sim.world
world.size = [1 * m, 1 * m, 1 * m]
```

```python
import gam_gate as gam
import gam_g4 as g4
import pathlib
import os
```

```python
# add a material database
sim.add_material_database(paths.data / 'GateMaterials.db')

# units
m = gam.g4_units('m')
cm = gam.g4_units('cm')
MeV = gam.g4_units('MeV')
Bq = gam.g4_units('Bq')
mm = gam.g4_units('mm')

# change world size
world = sim.world
world.size = [1 * m, 1 * m, 1 * m]

# add a simple fake volume to test hierarchy
# translation and rotation like in the Gate macro
fake = sim.add_volume('Box', 'fake')
fake.size = [40 * cm, 40 * cm, 40 * cm]
fake.material = 'G4_AIR'
fake.color = [1, 0, 1, 1]
fake.rotation = Rotation.from_euler('x', 20, degrees=True).as_matrix()

# image
patient = sim.add_volume('Image', 'patient')
patient.image = paths.data / 'patient-4mm.mhd'
patient.mother = 'fake'
patient.material = 'G4_AIR'  # material used by default
patient.voxel_materials = [[-2000, -900, 'G4_AIR'],
                           [-900, -100, 'Lung'],
                           [-100, 0, 'G4_ADIPOSE_TISSUE_ICRP'],
                           [0, 300, 'G4_TISSUE_SOFT_ICRP'],
                           [300, 800, 'G4_B-100_BONE'],
                           [800, 6000, 'G4_BONE_COMPACT_ICRU']]
# or alternatively, from a file (like in Gate)
vm = gam.read_voxel_materials(paths.gate_data / 'patient-HU2mat-v1.txt')
vm[0][0] = -2000
assert vm == patient.voxel_materials
patient.voxel_materials = vm
# write the image of labels (None by default)
patient.dump_label_image = paths.output / 'test009_label.mhd'
```

```python
# create the simulation
sim = gam.Simulation()

# main options
ui = sim.user_info
ui.g4_verbose = False
ui.g4_verbose_level = 1
ui.visu = False
ui.random_engine = 'MersenneTwister'
ui.random_seed = 'auto'
ui.number_of_threads = 5

sim.add_material_database(paths.data / 'GateMaterials.db')

# units
m = gam.g4_units('m')
cm = gam.g4_units('cm')
MeV = gam.g4_units('MeV')
Bq = gam.g4_units('Bq')
mm = gam.g4_units('mm')

# change world size
world = sim.world
world.size = [1 * m, 1 * m, 1 * m]
```

```python
# add a simple fake volume to test hierarchy
# translation and rotation like in the Gate macro
fake = sim.add_volume('Box', 'fake')
fake.size = [40 * cm, 40 * cm, 40 * cm]
fake.material = 'G4_AIR'
fake.color = [1, 0, 1, 1]
fake.rotation = Rotation.from_euler('x', 20, degrees=True).as_matrix()

# image
patient = sim.add_volume('Image', 'patient')
patient.image = paths.data / 'patient-4mm.mhd'
patient.mother = 'fake'
patient.material = 'G4_AIR'  # material used by default
patient.voxel_materials = [[-2000, -900, 'G4_AIR'],
                           [-900, -100, 'Lung'],
                           [-100, 0, 'G4_ADIPOSE_TISSUE_ICRP'],
                           [0, 300, 'G4_TISSUE_SOFT_ICRP'],
                           [300, 800, 'G4_B-100_BONE'],
                           [800, 6000, 'G4_BONE_COMPACT_ICRU']]
# or alternatively, from a file (like in Gate)
vm = gam.read_voxel_materials(paths.gate_data / 'patient-HU2mat-v1.txt')
vm[0][0] = -2000
assert vm == patient.voxel_materials
patient.voxel_materials = vm
```

```python
# default source for tests
source = sim.add_source('Generic', 'mysource')
source.energy.mono = 130 * MeV
source.particle = 'proton'
source.position.type = 'sphere'
source.position.radius = 10 * mm
source.position.translation = [0, 0, -14 * cm]
source.activity = 10000 * Bq
source.direction.type = 'momentum'
source.direction.momentum = [0, 0, 1]

# cuts
c = sim.get_physics_user_info().production_cuts
c.patient.electron = 3 * mm
```

```python
# add dose actor
dose = sim.add_actor('DoseActor', 'dose')
dose.save = paths.output / 'test009-edep.mhd'
dose.mother = 'patient'
dose.size = [99, 99, 99]
dose.spacing = [2 * mm, 2 * mm, 2 * mm]
dose.img_coord_system = True
dose.translation = [2 * mm, 3 * mm, -2 * mm]
dose.hit_type = 'random'

# add stat actor
stats = sim.add_actor('SimulationStatisticsActor', 'Stats')
stats.track_types_flag = True
```

```python
# create the simulation
sim = gam.Simulation()

# main options
ui = sim.user_info
ui.g4_verbose = False
ui.g4_verbose_level = 1
ui.visu = False

# add a material database
sim.add_material_database(paths.data / 'GateMaterials.db')

# units
m = gam.g4_units('m')
cm = gam.g4_units('cm')
MeV = gam.g4_units('MeV')
Bq = gam.g4_units('Bq')
mm = gam.g4_units('mm')

# change world size
world = sim.world
world.size = [1 * m, 1 * m, 1 * m]

# add a simple fake volume to test h
# translation and rotation like in t
fake = sim.add_volume('Box', 'fake')
fake.size = [40 * cm, 40 * cm, 40 *
fake.material = 'G4_AIR'
fake.color = [1, 0, 1, 1]
fake.rotation = Rotation.from_euler(

# image
patient = sim.add_volume('Image', 'pa
patient.image = paths.data / 'patien
patient.mother = 'fake'
patient.material = 'G4_AIR' # materi
patient.voxel_materials = [[-2000, -9
                           [-900, -100, 'Lung'],
                           [-100, 0, 'G4_ADIPOSE_TISSUE_ICRP'],
                           [0, 300, 'G4_TISSUE_SOFT_ICRP'],
                           [300, 800, 'G4_B-100_BONE'],
                           [800, 6000, 'G4_BONE_COMPACT_ICRU']]
# or alternatively, from a file (like in Gate)
vm = gam.read_voxel_materials(paths.gate_data / 'patient-HU2mat-v1.txt')
vm[0][0] = -2000
assert vm == patient.voxel_materials
patient.voxel_materials = vm
# write the image of labels (None by default)
patient.dump_label_image = paths.output / 'test009_label.mhd'
```

```python
# default source for tests
source = sim.add_source('Generic', 'mysource')
source.energy.mono = 130 * MeV
source.particle = 'proton'
source.position.type = 'sphere'
source.position.radius = 10 * mm
source.position.translation = [0, 0, -14 * cm]
source.activity = 10000 * Bq
source.direction.type = 'momentum'
source.direction.momentum = [0, 0, 1]

# cuts
c = sim.get_physics_user_info().production_cuts
```

```python
# create G4 objects
sim.initialize()


# start simulation
sim.start()
```

```python
# print info
print(sim.dump_volumes())

# verbose
sim.apply_g4_command('/tracking/verbose 0')

# start simulation

sim.start()
```

Status

Still not equivalent to current GATE. Estimate : 30%

- **Available** (among others):
  - Time slicing, basics volumes, voxelized volumes, boolean volumes, repeaters
  - All G4 physics lists, production cuts in volume, spliting/Russian Roulette (partial)
  - Generic source, voxelized source, beta+ source (!), confine, motion source, GAN source
  - DoseActor, PhaseSpaceActor (root), Adder (digitizer), AcceptanceAngle

- **Not yet implemented** (among others):
  - STL volume
  - Tracking cuts, advanced cuts, Optical, Polarization, DNA,
  - Back to back gamma source, Y90 source
  - Digitizer modules, LET, TLE, ARF, FFD

Estimation: can reach 80% within 1-2 years. Last 5% will be very hard

Warning : "it is hard to make prediction, especially about the future" ;)

Wanna try ?

Set your Python environment (optional, but highly recommended)

pip install gam-gate

gam_gate_tests

See: https://github.com/OpenGATE/gam-gate

Docs https://gam-gate.readthedocs.io (incomplete, only started)

And tests/examples (> 50 !):

https://github.com/OpenGATE/gam-gate/tree/master/gam_tests/src

Code example

https://github.com/OpenGATE/gam-gate/blob/master/gam_tests/src/test004_simple.py

https://github.com/OpenGATE/gam-gate/blob/master/gam_tests/src/test009_voxels.py

Under the hoods

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

Folder gam_gate	User UI (initialisation)

Some information

- Not all Geant4 fct, classes are exposed to Python (can be extended)
- Object destruction policy C++ vs Python
  - If, C++ pointer, need to keep a point in Python side
- Not possible to create or start 2 G4 simulation in the same script
- Image management via ITK:
  - Accessible via both C++ and Py sides
  - If transfer: need to copy !
- Warning for actors and scorers
  - Need to comply to multithread (need lock on shared variables)
  - Need to comply to multi-run and timed simulation

Python side

- One main object : Simulation
- 4 sub-main concepts: volume source physics actors
- All users parameters like “dictionary” structure
- Stored in user_info object