New and improved features in GATE 10

Nils Krah, CREATIS, Centre Léon Bérard, Lyon

GATE scientific meeting May 2024, Orsay

New and improved features that I will talk about

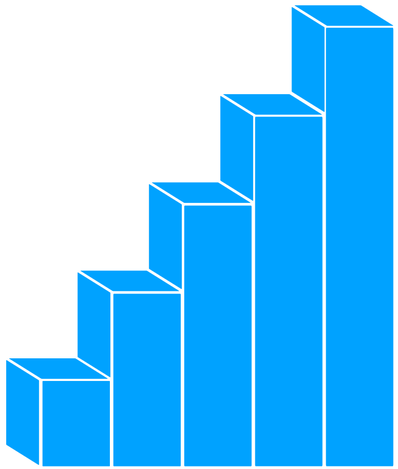
- Combine python and GATE
- Store simulation as JSON file
- Repeated volumes
- GATE in an interactive python terminal
- Dynamic parametrisations (e.g. moving geometry)
- Advanced actors
- Outlook: Digitizers, post-processors, HDF5

Combine python and GATE

Perform numerical calculations and run GATE simulations from the same script.

Example: construct a step phantom:

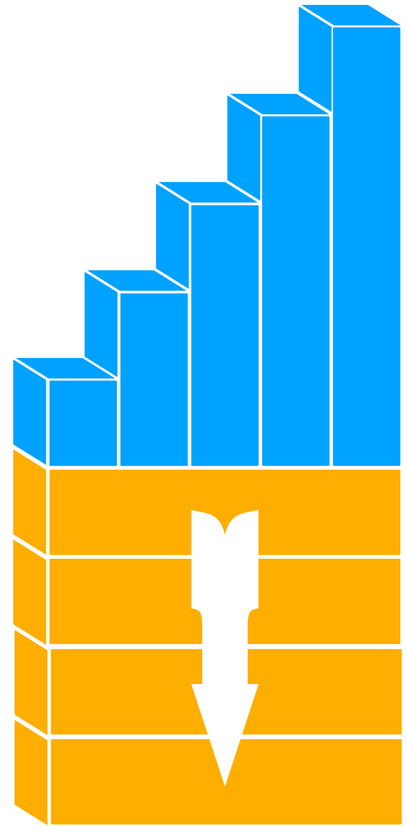
```
sim = gate.Simulation()
cm = gate.g4_units.cm
step_thickness = np.arange(2, 12, 2) * cm

for i, t in enumerate(step_thickness):
  b = sim.add_volume('Box', name=f"step_{i}")
  b.size = [2 * cm, 10 * cm, t]
  b.translation = [i * 2 * cm, 0, 0]

…geometry, physics, actors, sources,

sim.run()
```



Combine python and GATE

From previous example: successively add additional absorbers and re-run simulation.

```
sim = gate.Simulation()
cm = gate.g4_units.cm
step_thickness = np.arange(2, 12, 2) * cm
steps = []
for i, t in enumerate(step_thickness):
    b = sim.add_volume('Box', name=f"step_{i}")
    b.translation = [i * 2 * cm, 0, 0]
    steps.append(b)

…geometry, physics, actors, sources,

for a in np.arange(0, 10, 2) * cm:
    for step in steps:
        step.size = [2 * cm, 10 * cm, t + a]
    sim.output_dir = f"sim_absorber_{a}_cm"
    sim.run(start_new_process=True)
```



Store simulation as JSON

Rationale:

- A simulation python script is the interface via which you set-up and configure a simulation.
- You might run multiple simulations with the same script, each with a different configuration, e.g. different geometry, different physics.
- It would be nice to store the configuration of a simulation along with the results …
- … to use in data analysis
- … to re-run a simulation for later verification, e.g. in a clinical setting
- Dispatch the same simulation to servers by pushing configuration files.

Store simulation as JSON

Here is what you can do in GATE 10:

```
sim = gate.Simulation()
…
# set up the simulation
…
sim.store_json_archive = True
sim.json_archive_filename = “sim_with_super_strange_physics.json”

sim.run()
```

This will create a structured, human-readable text file in JSON format at the end of the simulation.

Caveat: Does not work for actors and sources so far.

Store simulation as JSON

Screenshot of example simulation JSON file:


```
{
    "user_info": {"name": "simulation"...},
    "object_type": "Simulation",
    "object_type_full": "<class 'opengate.managers.Simulation'>",
    "class_module": "opengate.managers",
    "i_am_a_gate_object": true,
    "volume_manager": {
        "user_info": {"name": "VolumeManager"...},
        "object_type": "VolumeManager",
        "object_type_full": "<class 'opengate.managers.VolumeManager'>",
        "class_module": "opengate.managers",
        "i_am_a_gate_object": true,
        "volumes": {
            "world": {"object_type": "BoxVolume"...},
            "rod": {"object_type": "TubsVolume"...},
            "waterbox_with_hole": {
                "user_info": {"name": "waterbox_with_hole"...},
                "object_type": "BooleanVolume",
                "object_type_full": "<class 'opengate.geometry.volumes.BooleanVolume'>",
                "class_module": "opengate.geometry.volumes",
                "i_am_a_gate_object": true
            },
            "patient": {"object_type": "ImageVolume"...}
        },
        "parallel_world_volumes": []
    },
    "physics_manager": {"object_type": "PhysicsManager"...}
}
```

Store simulation as JSON

You can also dump the JSON file without running the simulation:

```
sim = gate.Simulation()
…
# set up the simulation
…
sim.to_json_file(filename='simulation_that_has_not_run.json')
```

This will create a structured, human-readable text file in JSON format without actually running the simulation.

Reload simulation from JSON

You can also dump the JSON file without running the simulation:

```
sim = gate.Simulation()

sim.from_json_file('simulation_that_has_not_run.json')

sim.run()
```

This will read back the information from the JSON file and set-up the simulation objects as if they were defined in this script.

Caveat: Does not work for actors and sources so far.

Repeated volumes

Repeating identical volumes is very simple in GATE 10.

Example: construct a 1D array of box volumes, e.g. collimators in a linac

```
sim = gate.Simulation()
cm = gate.g4_units.cm

collimator = sim.add_volume('Box', name='collimator')
translation_list = []
for t in np.linspace(-10, 10, 20):
  translation_list.append([t * cm, 0, 0])

# translation_list is a list of 3-vectors
# the following line automatically creates repetitions of the volume
collimator.translation = translation_list

…geometry, physics, actors, sources,

sim.run()
```

GATE creates multiple Geant4 Physical Volumes for the same GATE volume.

GATE in an interactive python terminal

You can run and explore GATE in an interactive session, such as ipython, and dynamically access GATE objects, dump their properties, print help, etc.

help(gate.geometry.volumes.TubsVolume)

tubs = gate.geometry.volumes.TubsVolume(name='tubs')

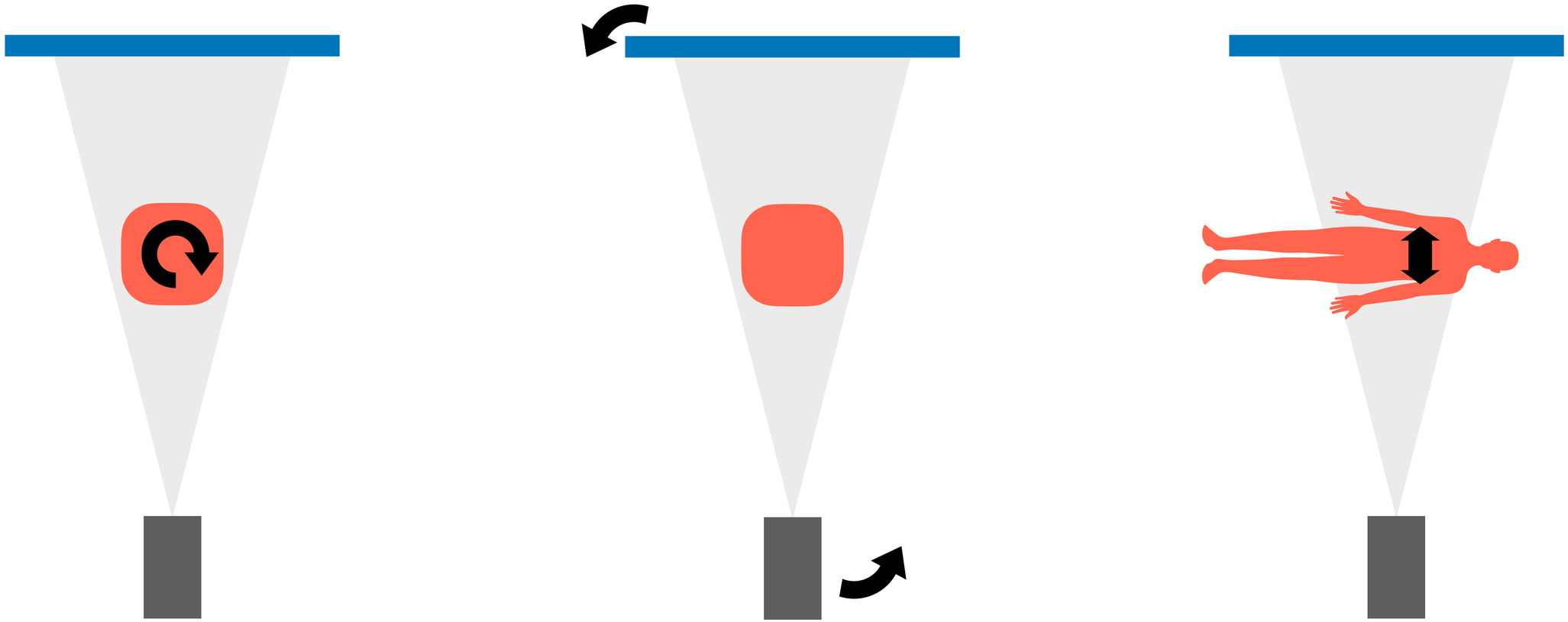
print(tubs)

tubs.set_ …tab completion … tubs.set_max_step_size

—> Demo tomorrow

Dynamic parametrisations

- Typical examples: moving geometry, moving phantom (e.g. breathing patient)



Dynamic parametrisations

Other imaginable parameters that might change during a simulation:

- Material properties
- Detector efficiency
- Source distribution, e.g. due to biological washout
- Source intensity or spectral distribution

GATE 10 provides architecture to implement such dynamic parametrisations.

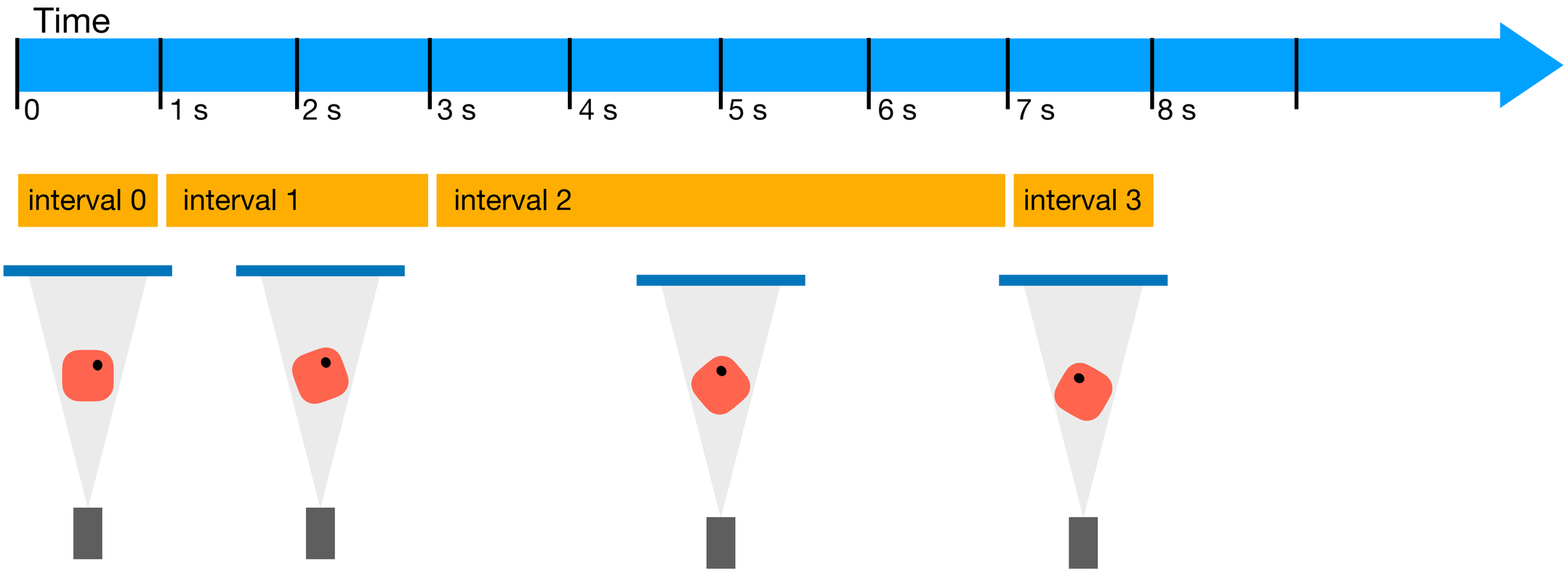
Dynamic parametrisations

How to make a simulation “dynamic”, e.g. evolve during the simulation?

- Short answer: You can’t do it!
- Long answer: **Yes, you can, but you need to discretize in time!**
- Split the simulation in time intervals and set different parameters for each interval.
- Example target rotation: Split simulation in 2 sec intervals and rotate 5 degrees further in between intervals.
- GATE 10 now implements a uniform interface for dynamic simulations.

Dynamic parametrisations

Simple code example: rotating target

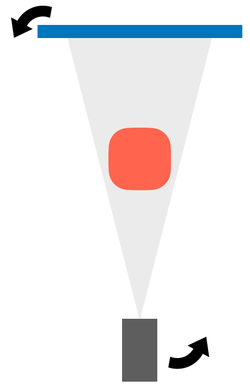


Dynamic parametrisations

Simple code example: rotating target

```
sim = gate.Simulation()
s = gate.g4_units.s
cm = gate.g4_units.cm

sim.run_timing_intervals = [(0, 1*s), (1*s, 3*s), (3*s, 7*s), (7*s, 8*s)]

waterbox = sim.add_volume("Box", name="waterbox")
waterbox.size = [20 * cm, 20 * cm, 20 * cm]
waterbox.translation = [-3 * cm, -2 * cm, -1 * cm]

rotations = []
for alpha [0, 20, 40, 60]:
    rotations.append(Rotation.from_euler("y", alpha, degrees=True).as_matrix())

waterbox.add_dynamic_parametrisation(rotation=rotations)
… (actors, source …)
sim.run()
```

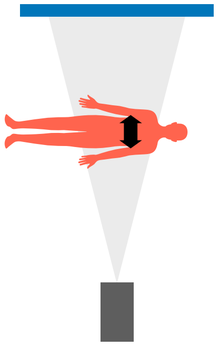
Dynamic parametrisations



Rotating gantry using helper function **get_transform_orbiting()**

```
sim = gate.Simulation()
…
nozzle = sim.add_volume("Box", “nozzle”)
detector = sim.add_volume("Box", “detector")

n = 10
interval_length = 3 * s
sim.run_timing_intervals
    = [(i * interval_length, (i + 1) * interval_length) for i in range(n)]
gantry_angles_deg = [i * 5 * deg for i in range(n)]

t, r =
    gate.geometry.utility.get_transform_orbiting(
        initial_position= nozzle.translation, axis="Y", angle_deg=gantry_angles_deg)
nozzle.add_dynamic_parametrisation(rotation=r, translation=t)

… analogous for the detector …
… (actors, source …)
sim.run()
```

Dynamic parametrisations



Example: dose simulation with 4D CT image

```
sim = gate.Simulation()
…
patient = sim.add_volume("Image", "patient")

# model sinusoidal breathing pattern and calculate corresponding time intervals:
times_cos_first_half = np.arccos(np.linspace(1, -1, 6))[:-1]
times_cos_second_half = np.pi * 2 - np.arccos(np.linspace(-1, 1, 6))
times_cos = np.concatenate((times_cos_first_half, times_cos_second_half))
times_cos /= 2 * np.pi  # normalize tp [0, 1) interval
interval_lengths = np.diff(times_cos)

sim.run_timing_intervals =
  [(t * sec, (t + q) * sec) for t, q in zip(times_cos[:-1], interval_lengths)]

paths_4d_ct = [f"4d_ct_phase_{10*i}.mhd" for i in range(10)]
patient.add_dynamic_parametrisation(image=paths_4d_ct)

… (actors, source …)
sim.run()
```

Almost ready: advanced actor output

- Unified output handling across actors
- Write to one common output directory per simulation
- Get and write actor output per run, e.g. in dynamic simulation
- Automatically merge actor output
- Intuitive access to actor output also when the simulation ran in a subprocess
- JSON dump of actor configuration

Outlook: digitizers, post-processors, HDF5

Observation:

- Many digitizers actually perform post processing operations on per-event data (singles).
- These digitizers do not need to iterate via the Geant4 event loop, but could iterate through events after the simulations.
- Digitizer C++ code is very complex -> difficult to contribute functionality.
- ROOT files are not easy to handle and many users convert to numpy arrays or similar structures in later processing.
- Disadvantage ROOT files: cannot write in-place in existing file

Outlook: digitizers, post-processors, HDF5

Idea 1:

- Switch to HDF5 format as alternative to ROOT files.
- HDF5 is able to handle large data.
- Easy to store meta data.
- Modern python HDF5 libraries available.
- In-place writing.
- Highly compatible with numpy.
- Very suitable for work with AI libraries.

Outlook: digitizers, post-processors, HDF5

Idea 2:

- Re-write part of digitizers as post-processors in python
- Construct a post-processor as a chain of pluggable processing units whose output provides the input to the next unit.
- Post-processors can be attached to actors to take their output as input to the processing chain.
- Post-processors can be run automatically as part of the simulation, or later based on actor output stored on disk.
- Should provide an easy framework within which new functionality can be contributed.

Outlook: digitizers, post-processors, HDF5

How would digitizers in the future GATE 10 look:

1. Keep as actors (digitizers): HitsCollectionsActor, AdderActor (groups hits into singles), PhaseSpaceActor
2. Potentially unify these three into one actor (they perform very similar operations)
3. Replace remaining digitizers by post-processors, e.g. blurring, spectral binning, spatial binning (projection)
4. Make HDF5 the default list-mode format
5. Keep ROOT as option

We are curious about your opinions

New and improved features that I talked about

- Combine python and GATE
- Store simulation as JSON file
- Repeated volumes
- GATE in an interactive python terminal
- Dynamic parametrisations (e.g. moving geometry)
- Advanced actors
- Outlook: Digitizers, post-processors, HDF5

Thanks for listening