Joint ICTP-IAEA Workshop on Monte Carlo Radiation Transport and Associated Data Needs for Medical Applications
28 October – 8 November 2024
ICTP, Trieste, Italy

Lecture 34

Automation of simulations

Reid Townson

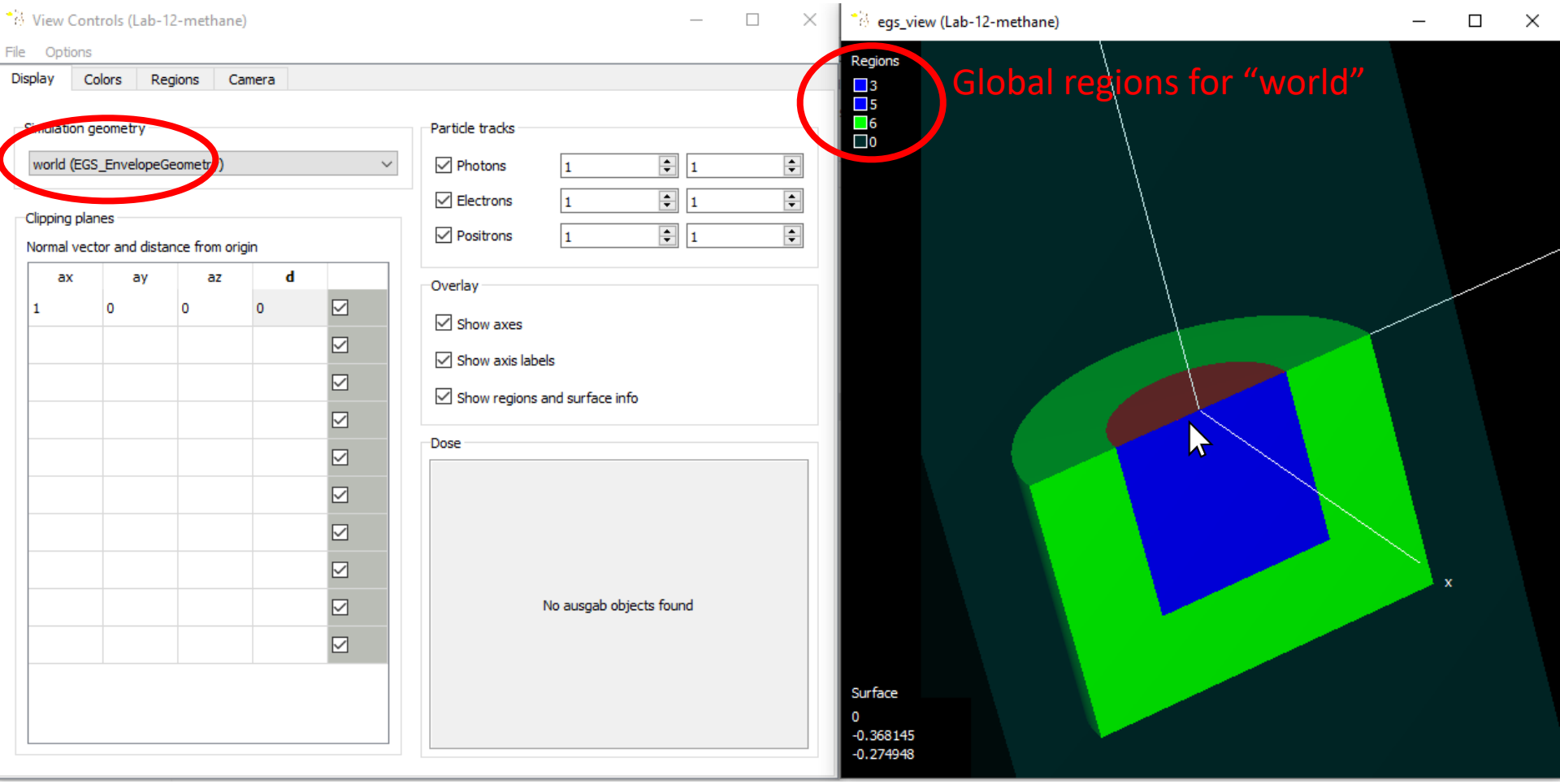
Metrology Research Centre
National Research Council Canada

Use labels for regions of interest

- The region numbering for complex geometries is tricky to follow
- If you add a new geometry, the number of everything else might change!
- To avoid needing to update all your dose scoring/ cavity/source regions of interest, use **labels**

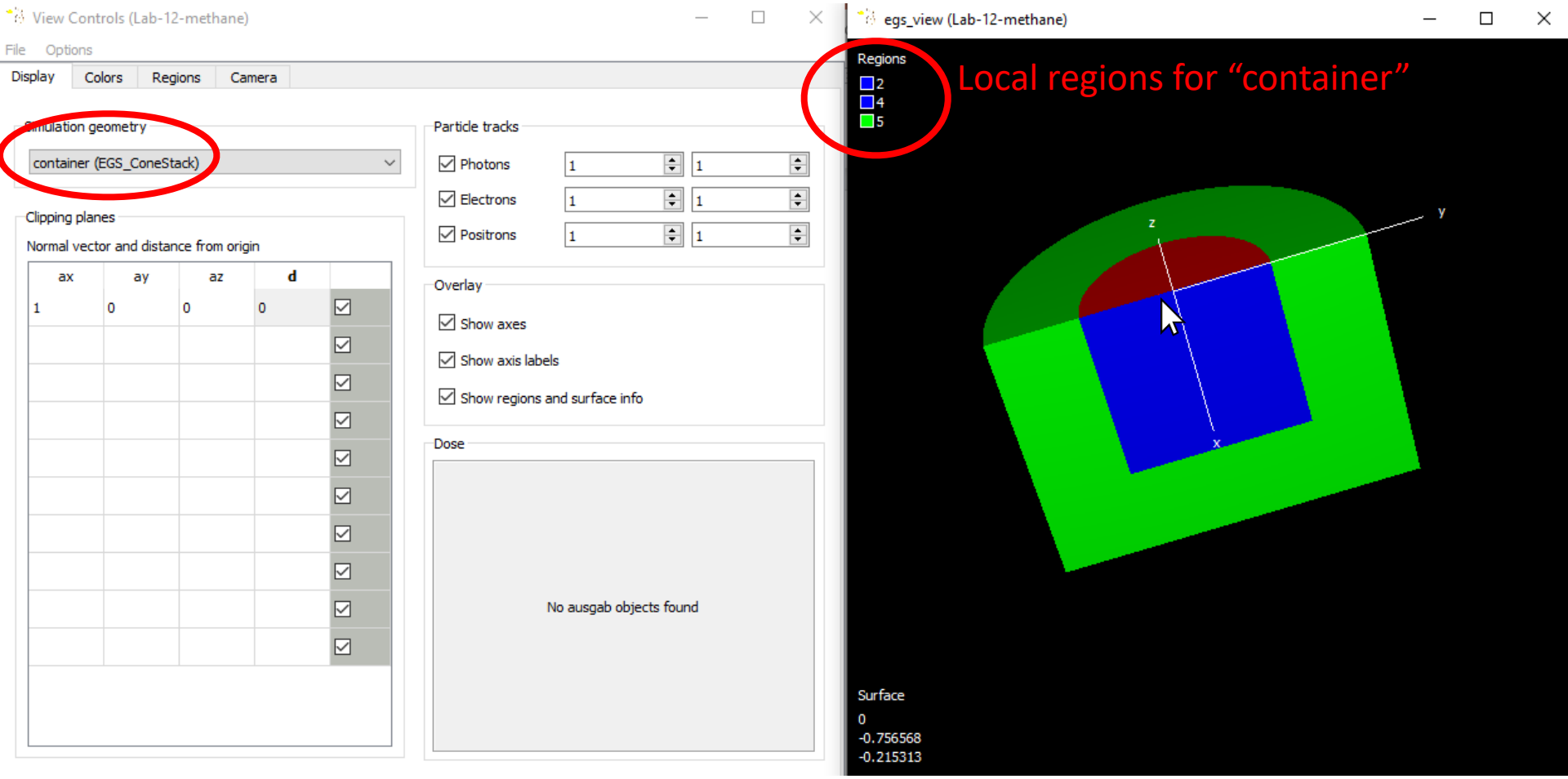
```
:start geometry:
        name = container
        library = egs_cones
        type = EGS_ConeStack
        … you know the rest…

        # Labels for regions of interest
        # These are local region numbers for this geometry
        set label = window 1
        set label = methane-top 2 4
:stop geometry:
```

Global region numbers might change



Local region numbers stay constant*

*Unless you change this geometry



- Set labels with local region numbers, and use the label instead where-ever region numbers are required

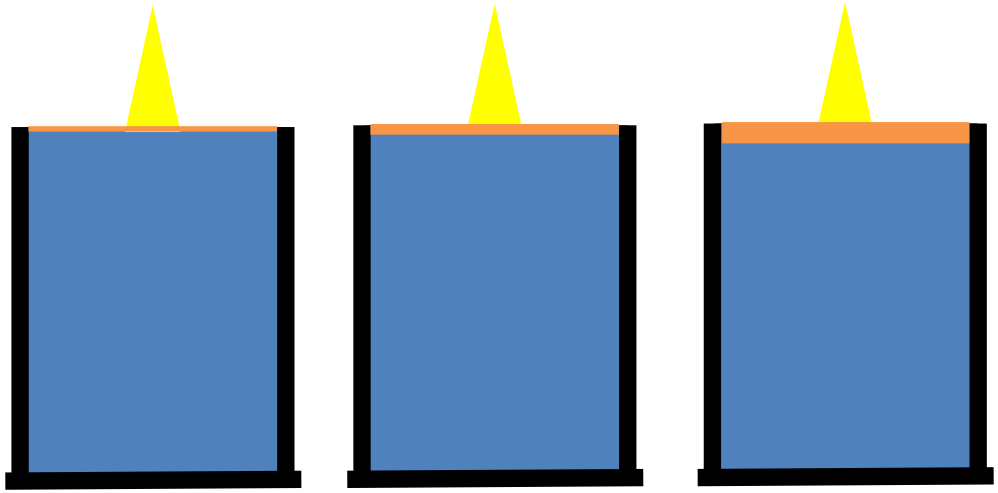
Use labels for regions of interest

- The labels can be used anywhere it asks for a list of regions
  - Multiple labels can be provided in the same list
  - You can mix labels and region numbers

```
:start ausgab object:
        library      = egs_dose_scoring
        name         = doseScoring
        region dose  = yes
        dose regions = methane-top
:stop ausgab object:
```

Monte Carlo simulations are ideal for relative comparisons

- Evaluating a parameter space is efficient and simple using MC simulation instead of experiment
- This is ideal for *designing* experiments
- Example: How thin should we make our detector window?



Automation allows for examining large parameter spaces

- It's possible to automate the simulation of thousands of simulations with small variations
- This may require large computing resources, efficient calculation conditions, and/or VRTs
- **Inside** `.egsinp` files, *input loops* provide automation
- **To generate** `.egsinp` files, use a scripting language like Python

Input loops in .egsinp files

- Input loops provide iterations over a parameter in .egsinp files
- Can repeat geometries at a different position, etc.

```
:start input loop:

    loop count = 5

    # Define a floating point loop variable (type=1) named var1
    # that takes values -45.0 + i*(-1) in the i'th loop.
    loop variable = 1 var1 -45.0 -1

     :start geometry:
        library = egs_planes
        type    = EGS_Zplanes
        name    = zplanes_-$(var1)cm
        positions = $(var1)
     :stop geometry:
:stop input loop:
```

Input loops in .egsinp files

- Loop variable format is:

```
loop variable = [type=0,1,2] [name] [offset]
[multiplier]
```

- Where type=0,1,2 corresponds to integer, float, and a list of strings, respectively
  - For a list a strings, use spaces to separate, e.g.

```
:start input loop:
    loop variable = 2 myVar Reid Fred Ernesto
    …etc…
```

  - Integer & float values are calculated as:

```
value=offset + multiplier*i
```

- Nesting loops is allowed!

Leverage input loops in egs_chamber

- egs_chamber has efficiency enhancing techniques tailored to repeated geometries
  - E.g. swapping out one material for another
- Use `TmpPhsp=1` to save a phase-space entering the "cavity" defined for the first "calculation geometry".
  - These particles are then re-used as the source for all the rest of the calculation geometries
  - Must be contained in the "cavity" for the phase-space

Scripting provides more flexibility

- Using a scripting language enables mathematical calculations, string processing and more
- A good strategy is to generate .egsinp files from a template, doing replacements of key values or sections
- This can also handle submitting/queueing many jobs
- Python is a popular choice

Python tutorial: replacements with %

- In Python, one can substitute local variables into a string using %

```
myValue = 10
variables = vars(self)

myString = “Here is the value: %(myValue)d” % variables


>>> myString
‘Here is the value: 10’
```

Python tutorial: replacements with %

- In Python, one can substitute local variables into a string using %

```
myValue = 10
variables = vars(self)

myString = "Here is the value: %(myValue)d" % variables


>>> myString
'Here is the value: 10'
```

Use d for digits, f for floats, and s for strings

Design a template input file

- Denote variables for replacement in your input file

```
:start run control:
    ncase = %(n_case)s
:stop run control:
```

- Use python to replace them and write a new file

```
template = open("myExample.template", "r")
content = template.read() % variables

egsinp = open("myExample.egsinp", "w")
egsinp.write(content)
```

Try our egs-rollout script

- The script handles arrays of parameters to automatically generate many input files
- For each project, you create a parameter script to define these values and arrays

```
> egs-rollout my-params.py
```

- Inside the `my-params.py` file:

```
param['template'] = 'myFile.template' # your template file
param['label'] = '_%(position).2f' # gets added to your filename

param['n_case'] = '1e6'

param['position'] = [0.15, 0.55] # an array of positions
```

  - Creates: `myFile_0.15.egsinp, myFile_0.55.egsinp`