FLUKA Geometry

FLUKA Advanced Course

Contents

- combinatorial geometry and format recalls
- bodies
  - generic quadric
  - transformations and rotation concepts
- regions
  - operators
  - problems and hints
  - parentheses
- lattice
- ancillary core routines
- dynamic objects
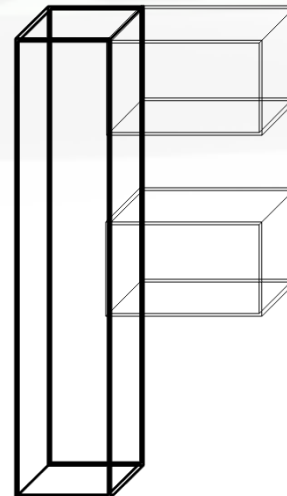- voxels (see dedicated tutorial on Medical Application on Friday)

Introduction

Principle of Combinatorial Geometry

Basic objects called bodies (such as cylinders, spheres, parallelepipeds, etc.) are combined to form more complex objects called regions

This combination is done using Boolean operations:

union, intersection, and subtraction

1 complex object

3 basic objects

Basic Concepts

Four concepts are fundamental in the FLUKA CG:

- **Bodies** - basic convex objects + infinite planes & infinite cylinders + generic quadric

- **Zones** - sub-region defined only with intersection and subtraction of bodies (used internally)

- **Regions** - are defined as boolean operations of bodies (union of zones)

- **Lattices** - duplication of existing regions (translated & rotated)

Geometry input format

- The input file format for the geometry is different from the one adopted anywhere else in FLUKA (i.e. the number and length of the input fields is different)

- Different formats can be used (due to backward compatibility)

  - **Fixed format**
    - Alignment is mandatory
    - Bodies and regions are identified by numbers (rather than names), this makes geometry editing more difficult

  - **Name based format**
    - **It is the recommended format**
    - **It is NOT the default!**

Name based format

It has many advantages, the main are:

- input parameter alignment is not necessary
- bodies and regions are identified by names
- possible to edit the input sequence without affecting the region description (e.g. inserting a new body)
- parentheses can be used to perform complex Boolean operations in the description of regions.

Name based format input is used for both body and region

It is activated

either by a GLOBAL command at the beginning of the input file

or putting SDUM = COMBNAME in the GEOBEGIN card

Input structure

FLUKA CG input must respect the following sequential order:

GEOBEGIN card
   VOXELS card   *(optional)*
   Geometry title (and reading format options)
   Body data
   $Start_transformation   *(optional)*

   Body data
   $End_transformation   *(optional)*

   Body data
   END card   *(automatically added by flair)*
   Region data

   END card   *(automatically added by flair)*
   LATTICE cards   *(optional)*
   Region volumes   *(optionally requested by a flag in the Geometry title, used together with **SCORE**)*
   GEOEND card

Cards having a * in column 1 are treated as comments

GEOBEGIN card

```
GEOBEGIN                          Log: ▼              Acc:              Opt: ▼
                                  Inp: ▼              Out: ▼            Fmt: COMBNAME ▼
        Title:
*...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....
GEOBEGIN                                              COMBNAME
     0      0
```

WHAT(1)   not used

WHAT(2)   set accuracy parameter – use with extreme care !

          → see next slide

WHAT(3)   logical unit for the geometry input.
          Geometry input can be read from file; should be >20.0

WHAT(4)   logical unit for the geometry output
          Log feature; should be >20.0 ; Default is standard output

WHAT(5)   Parenthesis optimization level

WHAT(6)   not used

SDUM   COMBNAME or COMBINAT

   COMBNAME selects name based format, COMBINAT fixed format

   Default: COMBINAT (!)

   Is overwritten by WHAT(5) of a possible GLOBAL card

Tracking accuracy

❑ FLUKA uses systematically double precision mathematic (i.e. 16 significant digits)

❑ GEOBEGIN's WHAT(2)*10^{-6} cm is the *absolute accuracy (AA)* requested for tracking and boundary identification

❑ The *relative accuracy (RA)* achievable in double precision is of the order of 10^{-14}-10^{-15}

❑ *AA* should be larger than *RA*L* , being *L* the largest coordinate value in the considered problem, i.e. the whole geometry size (The outer blackhole shell containing the system does not count)

❑ From FLUKA2011, instead of giving WHAT(2) the past default value of 0.0001, the code guesses *L* from the size of the second largest body in the geometry and sets WHAT(2) accordingly. Its value is reported in the output file and, if needed, can be retuned by the user.

Bodies

- Each body divides the space into two domains inside and outside. The outside part is pointed to by the normal on the surface.

- 3-character code of available bodies:
  - RPP: Rectangular parallelepiped
  - SPH: Sphere
  - XYP, XZP, YZP: Infinite half space delimited by a coordinate plane
  - PLA: Generic infinite half-space
  - XCC, YCC, ZCC: Infinite circular cylinder, parallel to coordinate axis
  - XEC, YEC, ZEC: Infinite elliptical cylinder, parallel to coordinate axis
  - RCC: Right circular cylinder
  - REC: Right elliptical cylinder
  - TRC: Truncated right angle cone
  - ELL: Ellipsoid of revolution
  - **QUA**:   Generic quadric surface

- Other bodies ARB, RAW, WED, BOX
  - don't use them, they cause sometimes rounding problems

Important Notes

- Whenever it is possible, the following bodies should be preferred:

> PLA, RPP, SPH, XCC, YCC, ZCC,
> XEC, YEC, ZEC, XYP, XZP, YZP, QUA

These make tracking faster, since for them extra coding ensures that unnecessary boundary intersection calculations are avoided when the length of the next step is smaller than the distance to any boundary of the current region.

- Always **use as many digits as possible** in the definition of the body parameters, particularly for body heights (RCC, REC), and for direction cosines of bodies with slant surfaces.

The free format [or the high-accuracy fixed format] should always be used in these cases.

Bodies input

The input for each body consists of:

- the 3-letter code indicating the body type (RPP, ZCC...)
- a unique "body name" (alphanumeric identifier, 8 character maximum, **case sensitive**)
- a set of geometrical quantities defining the body

Different items can extend over as many lines as needed (Maximum 132 characters per line accepted)

Different items have to be separated by one or more blanks, or by one of the separators , / ; :

All values are in cm!

Generic quadric: QUA

A QUA is the most generic quadric surface

It is defined by 10 coefficients in the following order:

$$A_{xx} \; A_{yy} \; A_{zz} \; A_{xy} \; A_{xz} \; A_{yz} \; A_x \; A_y \; A_z \; A_0$$

corresponding to the equation

$$A_{xx} x^2 + A_{yy} y^2 + A_{zz} z^2 + A_{xy} xy + A_{xz} xz + A_{yz} yz + A_x x + A_y y + A_z z + A_0 = 0$$

or $\begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} A_{xx} & A_{xy}/2 & A_{xz}/2 & A_x/2 \\ A_{xy}/2 & A_{yy} & A_{yz}/2 & A_y/2 \\ A_{xz}/2 & A_{yz}/2 & A_{zz} & A_z/2 \\ A_x/2 & A_y/2 & A_z/2 & A_0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = 0$ i.e. $\mathbf{r}^T M_{QUA} \, \mathbf{r} = 0$

For example:

`QUA EllHyper 0.25 1.0 -4.0 0.0 0.0 0.0 0.0 0.0 0.0 -1`

is an elliptic hyperboloid with axis equal to z

`QUA Cylinder 0.5 1.0 0.5 0.0 1.0 0.0 0.0 0.0 0.0 -4.0`

is an infinite circular cylinder of radius 2 with axis {z=-x,y=0}
(i.e. at -45° on the xz plane)

`QUA EllParab 0.25 1.0 0.0 0.0 0.0 0.0 0.0 0.0 -1.0 0.0`

is an elliptic paraboloid with axis equal to z

13
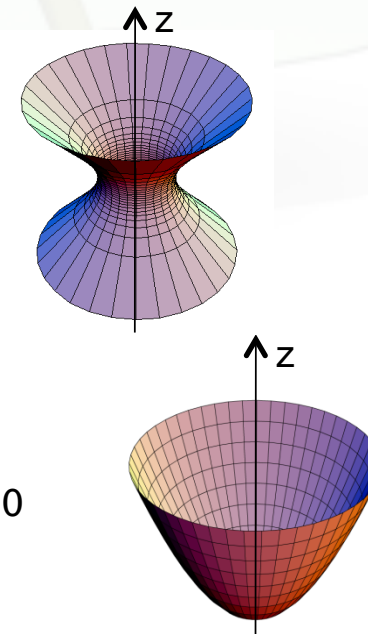
Geometry directives

Special commands enclosing body definition:

$Start_xxx

....

$End_xxx

where "xxx" stands for "expansion", "translat" or "transform"

They provide respectively a coordinate expansion/ reduction, a coordinate translation or a coordinate roto-translation of the bodies embedded between the starting and the ending directive lines.

◇ $start_transform	Trans: ▼		
◇ $end_transform			
◇ $start_expansion	f:		
◇ $end_expansion			
◇ $start_translat	dx:	dy:	dz:
◇ $end_translat			

Directives in geometry: expansion/reduction

➢ $Start_expansion ... $End_expansion

it provides a coordinate expansion (reduction) factor **f** for all bodies embedded within the directive

$Start_expansion 10.0
`SPH Sphere 5.0 7.0 8.0 50.0`
$End_expansion

$start_expansion		f: 10.0		
SPH Sphere	x: 5.0		y: 7.0	z: 8.0
	R: 50.0			
$end_expansion				

transforms a sphere of radius 50 centered in (+5,+7,+8)
into a sphere of radius 500 centered in (+50,+70,+80)

Putting the body in its quadric form

$$A_{xx} x^2 + A_{yy} y^2 + A_{zz} z^2 + A_{xy} xy + A_{xz} xz + A_{yz} yz + A_x x + A_y y + A_z z + A_0 = 0$$

or $\begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} A_{xx} & A_{xy}/2 & A_{xz}/2 & A_x/2 \\ A_{xy}/2 & A_{yy} & A_{yz}/2 & A_y/2 \\ A_{xz}/2 & A_{yz}/2 & A_{zz} & A_z/2 \\ A_x/2 & A_y/2 & A_z/2 & A_0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = 0$ i.e. $\mathbf{r^T} M_{QUA} \mathbf{r} = 0$

the expansion/reduction matrix is $T = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & f & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

and the transformed body equation is $\mathbf{r^T} (T^{-1})^T M_{QUA} T^{-1} \mathbf{r} = 0$

Directives in geometry: translation

➢ $Start_translat ... $End_translat

it provides a coordinate translation S_x, S_y, S_z for all bodies embedded within the directive

$Start_translat -5.0 -7.0 -8.0
SPH Sphere 5.0 7.0 8.0 50.0
$End_translat
transforms a sphere of radius 50 centered in (+5,+7,+8)
into a sphere of radius 50 centered in (0,0,0)

$start_translat	dx: -5.0	dy: -7.0	dz: -8.0
●SPH Sphere	x: 5.0	y: 7.0	z: 8.0
	R: 50.0		
$end_translat			

the translation matrix is $T = \begin{bmatrix} 1 & 0 & 0 & S_x \\ 0 & 1 & 0 & S_y \\ 0 & 0 & 1 & S_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Directives in geometry: roto-translation

➢ $Start_transform ... $End_transform

it applies a pre-defined (via ROT-DEFI) roto-translation to all bodies
embedded within the directive

```
ROT-DEFI , 201.0, 0., +116.5650511770780, 0., 0., 0., cylrot
$Start_transform cylrot
QUA Cylinder 0.5 1.0 0.5 0.0 1.0 0.0 0.0 0.0 0.0 -4.0
$End_transform
```
transforms an infinite circular cylinder of radius 2 with axis {x=-z,y=0}
into an infinite circular cylinder of radius 2 with axis {x=z/3,y=0} (**clockwise rotation**)

the roto-translation matrix is $T = \begin{bmatrix} & & & S_x \\ & R & & S_y \\ & & & S_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$

$T^{-1} = \begin{bmatrix} & & & -(R^{-1}S)_x \\ & R^{-1} & & -(R^{-1}S)_y \\ & & & -(R^{-1}S)_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$

- it allows to rotate a RPP avoiding the use of the deprecated BOX !

- note that also the **inverse** transformation can be used T^{-1}

$Start_transform **-**cylrot

ROT-DEFIni

The ROT-DEFIni card defines roto-translations that can be applied, in addition to bodies, to i. USRBIN & EVENTBIN and ii. LATTICE. It transforms the position of the tracked particle i. before scoring with respect to the defined binning or ii. into the prototype with the order:

- First applies the translation
- followed by the rotation on the azimuthal angle
- and finally by the rotation on the polar angle.

$$X_{new} = M_{polar} \times M_{az} \times (X + T)$$

WHAT(1): assigns a transformation index and the corresponding rotation axis

$$I + J * 100 \qquad or \qquad I * 1000 + J$$

I = index of rotation   (WARNING: NOTE THE SWAP OF VARIABLES)

J = rotation with respect to axis (1=X, 2=Y, 3=Z)

WHAT(2): Polar angle of the rotation ($0 \leq \vartheta \leq 180^o$ degrees)

WHAT(3): Azimuthal angle of the rotation ($-180 \leq \varphi \leq 180^o$ degrees)

WHAT(4), WHAT(5), WHAT(6) = X, Y, Z offset for the translation

SDUM: Optional (but recommended) name for the transformation

ROT-DEFI	Id: 1	Axis: Z ▼	Name: 1tra
	Polar: 0.0	Azm:	
	Δx:	Δy:	Δz: -10.0

18

Identifying rotation angles

Let's define the orientation of a body in the space
by a system of 3 orthogonal vectors **i', j', k'** , whose coordinates
are expressed with respect to the fixed reference frame X,Y,Z

Then $[\ i'\ \ j'\ \ k'\ ] = \begin{bmatrix} c_1 c_3 - c_2 s_1 s_3 & -c_1 s_3 - c_3 c_2 s_1 & s_2 s_1 \\ c_2 c_1 s_3 + c_3 s_1 & c_1 c_2 c_3 - s_1 s_3 & -c_1 s_2 \\ s_3 s_2 & c_3 s_2 & c_2 \end{bmatrix} \begin{bmatrix} X_i \\ Y \\ Z \end{bmatrix}$ (in the ZXZ convention)

where $c_1 = \cos(\psi)$ $c_2 = \cos(\theta)$ $c_3 = \cos(\Phi)$ $s_1 = \sin(\psi)$ $s_2 = \sin(\theta)$ $s_3 = \sin(\Phi)$
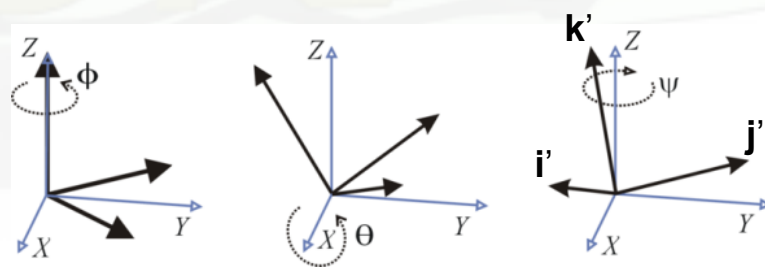
here $\Phi = 45^\circ$ $\theta = 30^\circ$ $\psi = -60^\circ$

The obtained Euler angles can be input as azimuthal angle of
three consecutive rotations (ROT-DEFI)

Directives in geometry: warnings

➢ **$Start_expansion** and **$Start_translat** are applied when reading the geometry → no CPU penalty (the concerned bodies are transformed once for ever at inizialization)

$Start_transform is applied runtime → some CPU penalty

➢ One can **nest** the different directives *(at most one per type!)* but, no matter the input order, the adopted sequence is always the following:

$Start_transform StupiRot

$Start_translat -5.0 -7.0 -8.0

$Start_expansion 10.0

```
QUA WhatIsIt +1.0 +1.0 +1.0 0.0 0.0 0.0 -10.0 -14.0 -16.0 -2362.0
```

$End_expansion

$End_translat

$End_transform

➢ Directives are not case sensitive (whereas roto-translation names are)

Regions (I)

Input for each region starts on a new line and extends on as many continuation lines as are needed. It is of the form:

REGNAME NAZ boolean-zone-expression | boolean-zone-expression | ...

- "*boolean-zone-expression*" is a sequence of one or more body names preceded by the operators **+** (intersection) or **−** (complement or subtraction). Several zone expressions can be combined by the union operator **|** . (A single boolean-zone-expression is admitted).
The operator precedence sequence is: first parentheses (see later), then **+** and **−** , last **|** .

- *REGNAME* is the region name (an arbitrary unique alphanumeric character string chosen by the user). The region name must begin by an alphabetical character and must not be longer than 8 characters.

Regions (II)

- **NAZ** (Number of Adjacent Zones) is a rough estimate of the number of zones a particle can enter when leaving the current region zones (5 by default). What actually matters is the NAZ sum over all regions, defining the size of the *contiguity list*

- While tracking, the program searches in the contiguity list for the neighbor zones of each zone. If the zone is not yet in the list, the whole geometry is scanned and it is added to the list with its neighbor zones

- When the NAZ limit is reached (i.e. the list is full) the code prints a warning: GEOMETRY SEARCH ARRAY FULL. This is not lethal: the calculation continues but with a reduced efficiency

- If you have more than 1000 regions, you must issue a GLOBAL card putting in WHAT(1) a higher limit (not beyond 10000)

Operators

Regions are defined as **combinations of bodies** obtained by boolean operations:

	Union	**Subtraction**	**Intersection**
Named based	|	−	+
Fixed	OR	−	+
Mathematically	∪	−	∩

Regions are not necessarily simply connected (they can be made as the union of two or more non contiguous or partially overlapping zones) but must be of **homogeneous material composition**.

Zones must be finite.

Regions Boolean expression

- **zones** are defined by intersections and/or subtraction of **bodies**, each body being preceded by its + or – sign
(+ in the first body not required)

- **|** (OR) operators are used to join **zones**

- In evaluating the expressions, the highest operator precedence is given to **parentheses**, followed by **+, -** and the **|** operator

- Parentheses are available in name based format (use with care!)

- If one line is not sufficient, any number of continuation lines can be added

- Blanks are ignored

Meaning of + and - operators

If a body number (or name) appears in a zone description preceded by a **+** operator, it means that the zone being described is wholly contained inside the body.

If a body number appears in a zone description preceded by a **–** operator, it means that the zone being described is wholly outside the body.

Zones must be finite: normally in the description of each zone and hence of each region the symbol **+** must appear at least once.

Meaning of the | (OR) operator

The **|** (or OR) operator is used as a Boolean **union** operator in order to combine **zones** (**sub regions** partially overlapping or not).

Zones are formed as body intersections or subtractions as previously explained

The **regions** are formed by the union of these **zones**.

Examples:

```
Ground 5 | +Body9 | +Body15 | +Body1 | +Body8 -Body2 | +Body8 -Body3
*          <- 1st -><-  2nd -><- 3rd -><----  4th ----><----  5th --->
* (continuation line)
         | +Body8 +Body18  | +Body12 -Body10 -Body11 -Body13 -Body14
*           <-----  6th -----><------- 7th and last zone     -------->
```

Illustration of the **+** and **−** operators
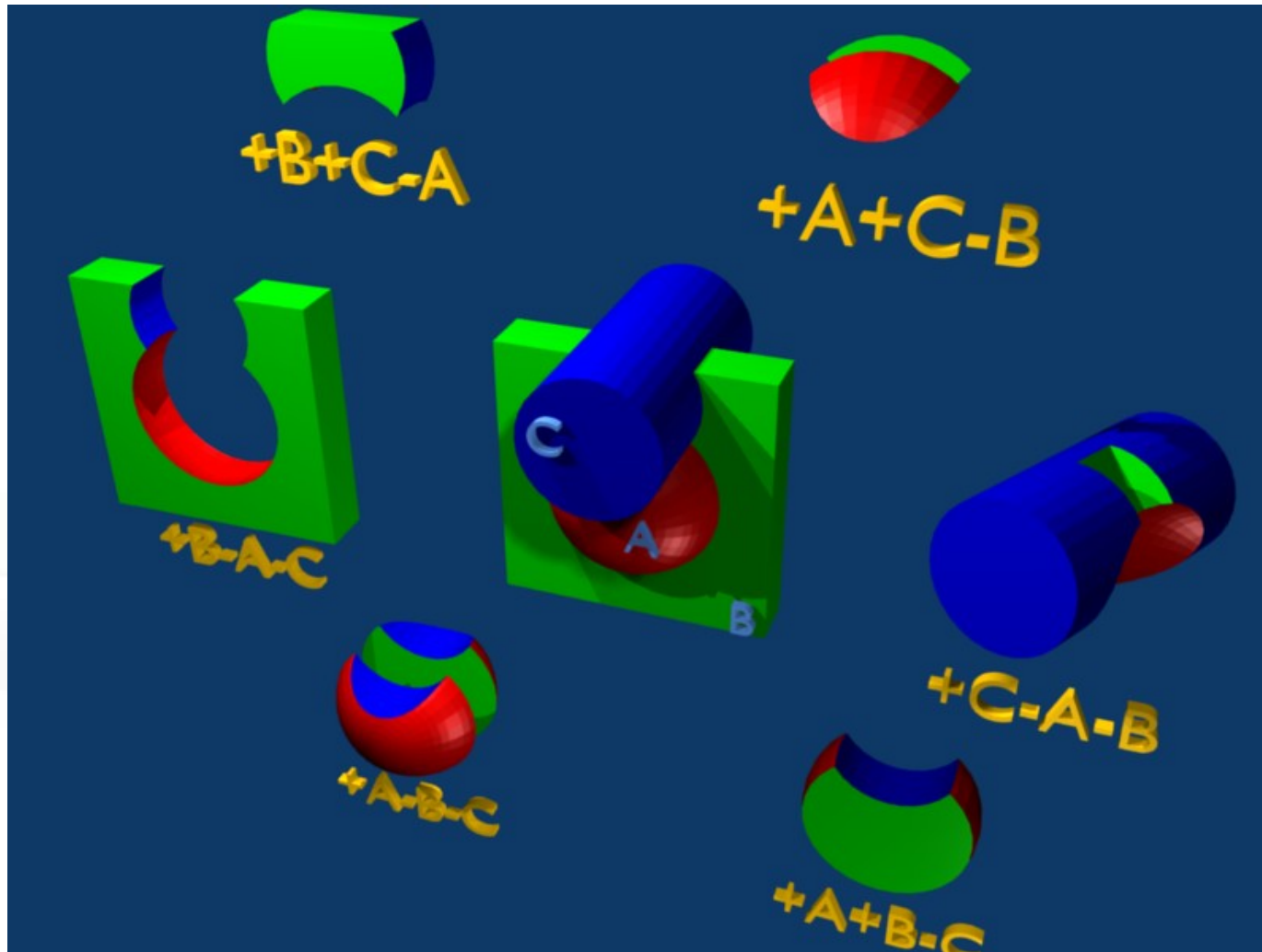
The Blackhole

To avoid infinite tracking the particles must be stopped somewhere. This has to be insured by the user by defining a region surrounding the geometry and assigning the material **BLCKHOLE** to it.

All particles that enter the blackhole are absorbed (they disappear and are flagged as escaping). Further blackhole regions can be defined by the user if necessary.

The blackhole is the outermost boundary of the geometry. Inside the blackhole region:

Each point of space must belong to one and only one region!

Geometry Example "F" shaped target

Material: Copper

Surrounded by vacuum

Black hole: spherical shell from 1000 – 10000 cm

1 cm thick in x direction
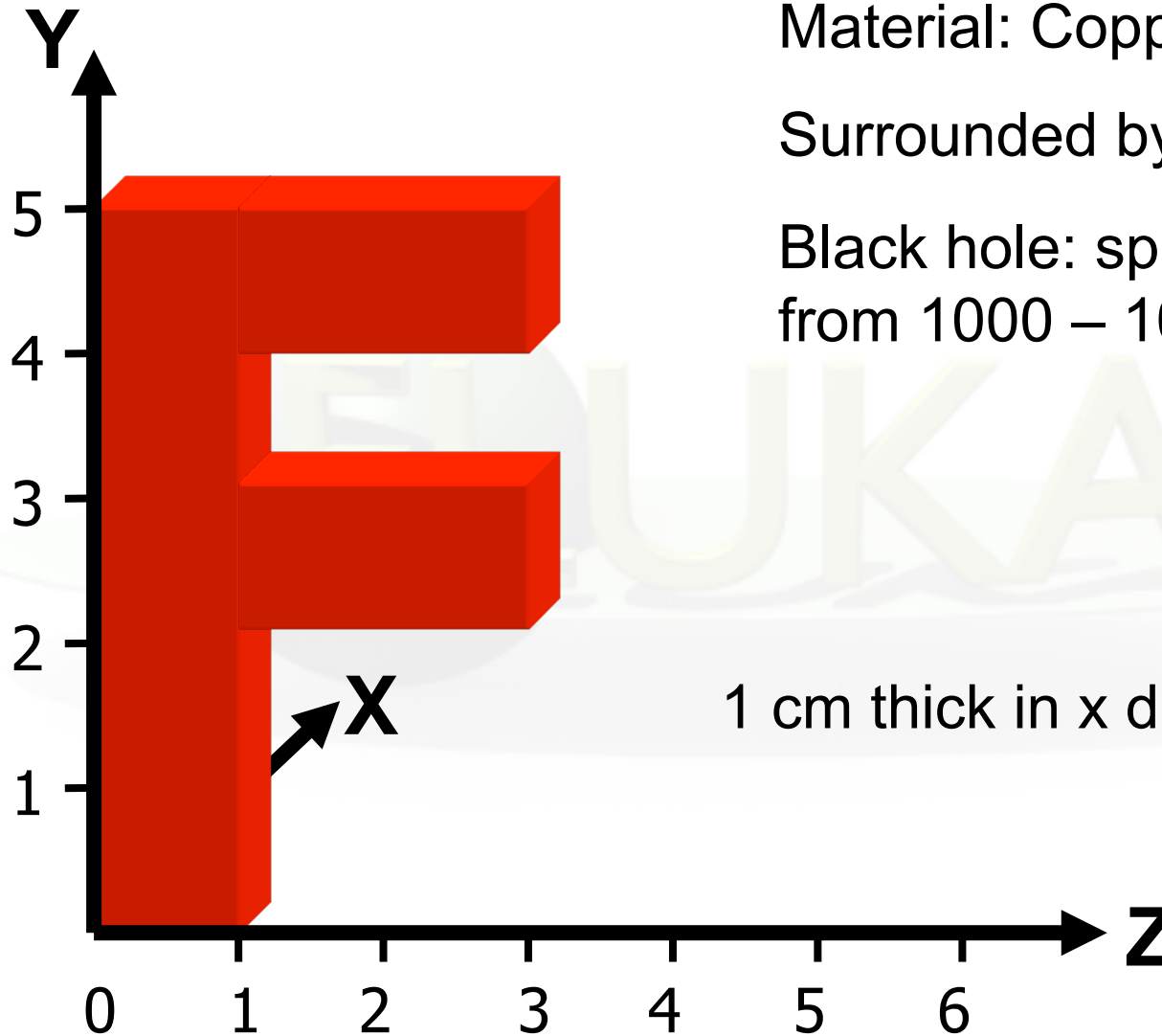
Geometry example "F": Bodies

Several possibilities for bodies:
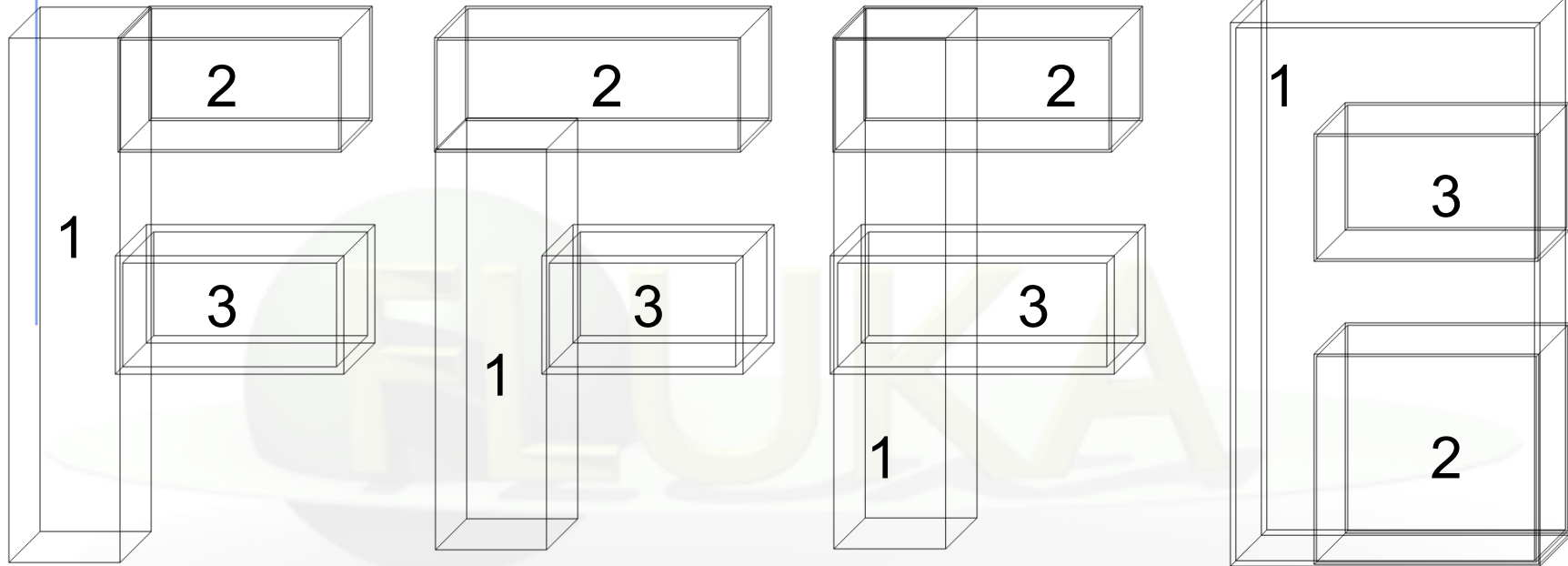
(A) 3 bodies   (B) 3 bodies   (C) overlapping   (D) subtraction

The 4 options are equivalent. We will use C.

Geometry example "F": REGIONS
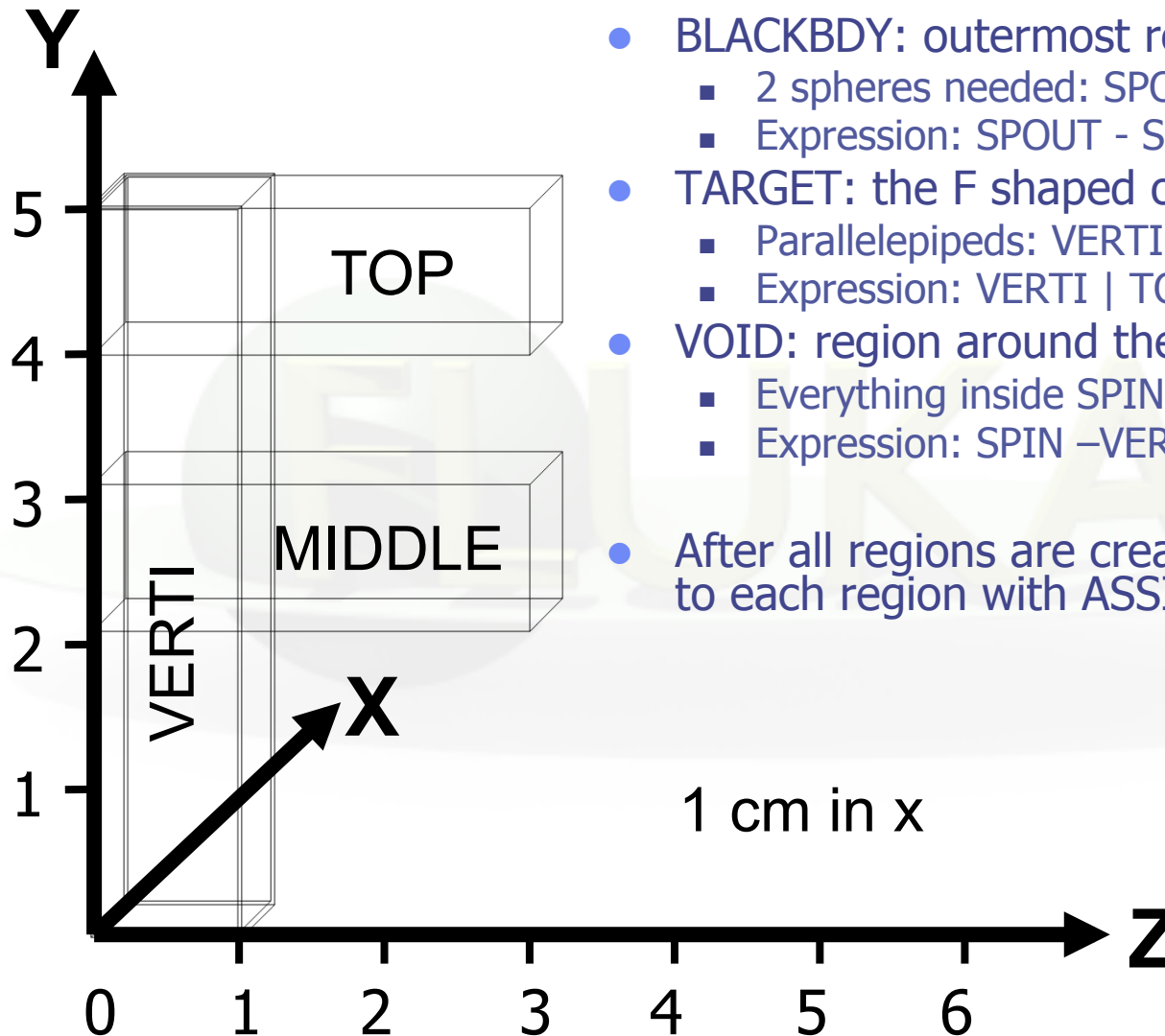
- **BLACKBDY**: outermost region like a shell
  - 2 spheres needed: SPOUT, SPIN
  - Expression: SPOUT - SPIN
- **TARGET**: the F shaped object
  - Parallelepipeds: VERTI, TOP, MIDDLE
  - Expression: VERTI | TOP | MIDDLE
- **VOID**: region around the target
  - Everything inside SPIN that is not target
  - Expression: SPIN –VERTI – TOP –MIDDLE

- After all regions are created: Assign material to each region with ASSIGNMA

Geometry example "F": input file

- ```
GEOBEGIN
COMBNAME
      0   0   The copper F
SPH SPIN   0.0 0.0 0.0 1000.
SPH SPOUT   0.0 0.0 0.0 10000.
RPP VERTI   0.0 1. 0.0 5. 0.0 1.
RPP TOP   0.0 1. 4. 5. 0.0 3.
RPP MIDDLE   0.0 1. 2. 3. 0.0 3.
END
* Black hole
BLKBODY   5 +SPOUT -SPIN
* Void around
VOID   5 +SPIN -TOP -VERTI -MIDDLE
* Target
TARGET   5 TOP | VERTI | MIDDLE
END
GEOEND
ASSIGNMA   BLCKHOLE   BLKBODY
ASSIGNMA   VACUUM   VOID
ASSIGNMA   COPPER   TARGET
```

# Geometry example "F": input Flair



| GEOBEGIN | | Log: ▼ | Acc: | Opt: ▼ |
|---|---|---|---|---|
| | | Inp: ▼ | Out: ▼ | Fmt: COMBNAME ▼ |
| | Title: The copper F | | | |
| SPH | SPOUT | x: 0 | y: 0 | z: 0 |
| | | R: 10000 | | |
| SPH | SPIN | x: 0 | y: 0 | z: 0 |
| | | R: 1000 | | |
| RPP | VERTI | Xmin: 0 | Xmax: 1 | |
| | | Ymin: 0 | Ymax: 5 | |
| | | Zmin: 0 | Zmax: 1 | |
| RPP | TOP | Xmin: 0 | Xmax: 1 | |
| | | Ymin: 4 | Ymax: 5 | |
| | | Zmin: 0 | Zmax: 3 | |
| RPP | MIDDLE | Xmin: 0 | Xmax: 1 | |
| | | Ymin: 2 | Ymax: 3 | |
| | | Zmin: 0 | Zmax: 3 | |

Black hole

| REGION | | Name: BLKBODY | Neigh: 5 | Volume: |
|---|---|---|---|---|
| | Expr: +SPOUT -SPIN | | | |

Void around

| REGION | | Name: VOID | Neigh: 5 | Volume: |
|---|---|---|---|---|
| | Expr: +SPIN -TOP -VERTI -MIDDLE | | | |

Target

| REGION | | Name: TARGET | Neigh: 5 | Volume: |
|---|---|---|---|---|
| | Expr: TOP |VERTI |MIDDLE | | | |
| GEOEND | ▼ | | | |

..+....1....+....2....+....3....+....4....+....5....+....6....+....7..

| ASSIGNMA | Mat: BLCKHOLE ▼ | Reg: BLKBODY ▼ | to Reg: ▼ |
|---|---|---|---|
| | | Step: | Field: ▼ |
| ASSIGNMA | Mat: VACUUM ▼ | Reg: VOID ▼ | to Reg: ▼ |
| | | Step: | Field: ▼ |
| ASSIGNMA | Mat: COPPER ▼ | Reg: TARGET ▼ | to Reg: ▼ |
| | | Step: | Field: ▼ |

# Geometry Errors

- Problem space not enclosed by a black body region

- Never start a primary particle along a surface. You could  get a geometry error even if the geometry is correct because FLUKA cannot determine the region

- Precision errors

- Lattice replica ←→ basic cell mismatch

# Preventing precision errors

Ideal world
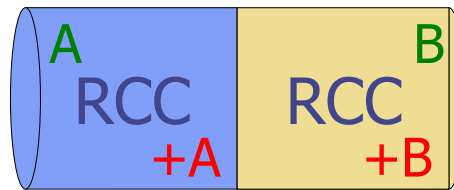
Computation world

A
B
RCC
RCC
+A
+B
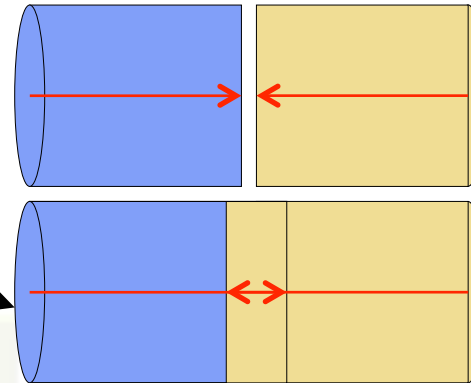
Avoid touching surfaces!

When floating point operations are involved

- Use cutting planes instead

B PLA

A
RCC
+A+B    +A-B
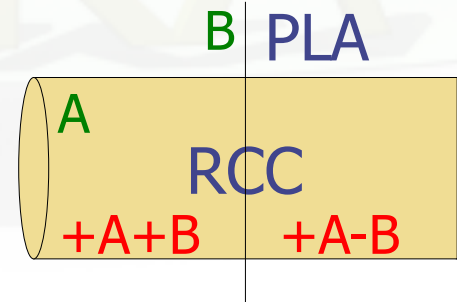
- Or force partial overlap of bodies

A ———————— B

A-B    +B
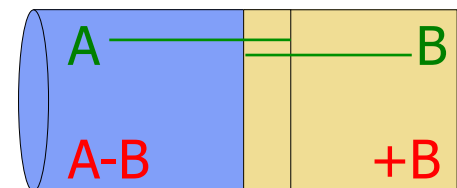
# Geometry error messages

During execution the code always needs to know the region where a particle is located at every step.

- The program will stop only if a particle's position does not belong to any region.
  It will issue an error message on the .err file with the particle position.

- IMPORTANT! It will not stop if a particle's position belongs to more than one region. It will accept the first region it finds but results will be unreliable!!
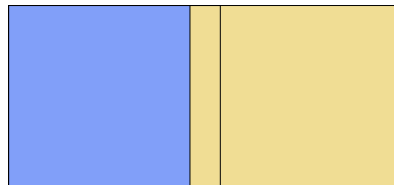
# Debugging tools

- GEOEND card with the DEBUG option

- Error messages during simulation in the .err file

- Geometry plotting by Flair
    (automatically invoking the PLOTGEOM card)

- **FLAIR Geometry Editor** (very powerful! See the dedicated lecture)

- SimpleGeo

# Native debugger (I)

GEOEND card activates the FLUKA geometry debugger.

Detects both undefined or multiple defined points in a selected X,Y,Z mesh

Two cards are needed

First card

| WHAT(1)=$X_{max}$ | WHAT(2)=$Y_{max}$ | WHAT(3)=$Z_{max}$ |
|---|---|---|
| WHAT(4)=$X_{min}$ | WHAT(5)=$Y_{min}$ | WHAT(6)=$Z_{min}$ |
| SDUM = DEBUG | | |

Second Card

| WHAT(1)=Nx | WHAT(2)=Ny | WHAT(3)=Nz |
|---|---|---|
| SDUM = & | | |

```
GEOEND  Xmax   Ymax   Zmax  Xmin  Ymin  Zmin  DEBUG

GEOEND  Nx     Ny     Nz                            &
```

GEOEND	DEBUG ▼		
Xmin:		Xmax:	NX:
Ymin:		Ymax:	NY:
Zmin:		Zmax:	NZ:

WARNING!

The program stops if too many errors are found

A message will be issued on the output unit

38

Native debugger (II)

- If no error is found, no .err file will be created.

- REMINDER: If the debugger doesn't find any error it doesn't mean that the geometry is error free!

- Errors will be listed in the .err file in the form:

  **** Lookdb: Geometry error found ****
  **** The point: -637.623762 -244.554455 -96.039604 ****

  - Point is contained in more than one region

  **** is contained in more than 1 region ****
  **** (regions: 6 7) ****

  - Not contained in any region

  **** is not contained in any region

- Exploit the geometry symmetry asking for 2D scans on planes

- Scan only the problematic areas

- Adopt a step length that prevents from ending up on "special" points (i.e. boundaries)

User optimization in region definition

A zone involving many bodies increases the tracking time, since the exiting conditions imply a respective number of checks

1731 holes

slices with <50 holes

one zone with MANY bodies:
large CPU time!

many zones with much less
bodies each: significant benefit

...ideally large zones (far boundaries) with few bodies (!)

$\Longrightarrow$ prefer overlapping zones

Parentheses

Parentheses are grouping together combinations of bodies.
Parentheses can be used in name based format only.

Examples:
```
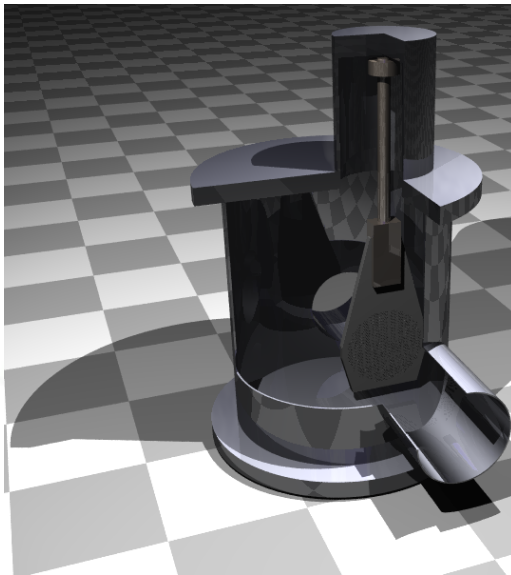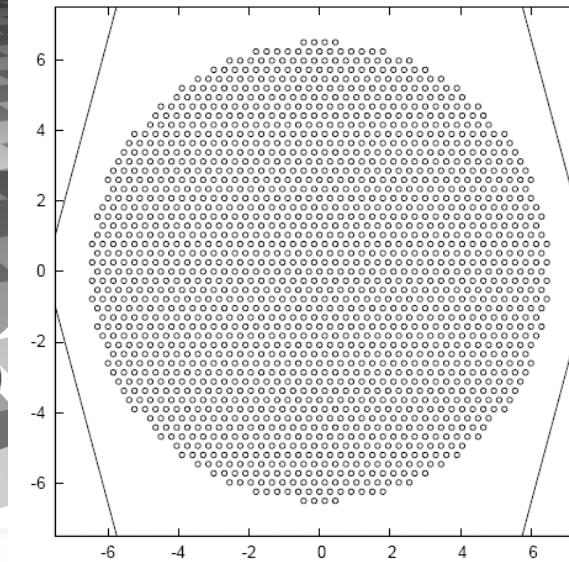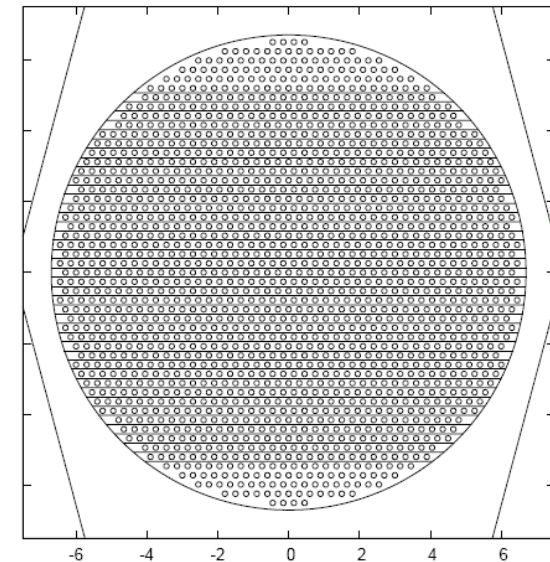*...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
* Subtract from body2 regions regR03, regR04, regR05
regV02 5  +body2 - (+body4  -body3)
         - (+body6  -body5 |+body8 -body7) - body9 -body10
regR03 5 +body4  -body3
regR04 5 +body6  -body5 |+body8 -body7
regR05 5 +body9 | +body10
```

Nested parentheses are supported, however:

parentheses should be used with care since their expansion can generate a quickly diverging amount of terms.

A partial optimization is performed on planes (aligned with the axes) and bounding boxes only

Parenthesis Expansion (I)

- Parentheses expansion is almost like converting from product of sums to sum of products

- Product operators are: **+/-**, Sum operator: **|**

- The final result will be an expression in the normal form. Unions of all possible combinations of the bodies in the expression!

- Initially the code removes all repeated terms:

A + A	= A
A − A	= $\varnothing$
expA | expA	= expA

Parenthesis Expansion (II)

Geometrical optimization can drastically reduce the number of zones

1. Elimination of same type of planes (XYP, XZP, YZP) inside a zone (product)

2. Optimization of zones based on the bounding boxes of the bodies.

   - Infinite objects have an infinite bounding box on some of the dimensions i.e. XYP, ZCC etc.

   - PLAnes do not have a bounding box

   - For each zone after the expansion, if the intersection of the bounding boxes is empty the zone is discarded

Lattice

FLUKA geometry has *replication* (lattice) capabilities

Only one *level is implemented* (no nested lattices are allowed)

- The user defines lattice positions in the geometry and provides transformation rules from the lattice to the prototype region:
  1. in the input with the ROT-DEFI card
  2. in a subroutine (lattic.f)

The lattice identification is available for scoring

Transformations include:

Translation, Rotation and Mirroring (the last only through routine).

WARNING:

Do not use scaling or any deformation of the coordinate system

Lattice

- The regions which constitute the elementary cell *(prototype)* to be replicated, have to be defined in detail

- The Lattices *(replicas/containers)* have to be defined as "empty" regions in their correct location.
  WARNING: The lattice region should map exactly the outer surface definition of the elementary cell.

- The lattice regions are declared as such with a LATTICE card at the end of the geometry input

- In the LATTICE card, the user also assigns lattice names/ numbers to the lattices. These names/numbers will identify the replicas in all FLUKA routines and scoring

- Several basic cells and associated lattices can be defined within the same geometry, one LATTICE card will be needed for each set

- Non-replicas carry the lattice number **0**

- Lattices and plain regions can coexist in the same problem

LATTICE card

After the Regions definition and before the GEOEND card the user can insert the LATTICE cards

- WHAT(1), WHAT(2), WHAT(3)
  Container region range (from, to, step)

- WHAT(4), WHAT(5), WHAT(6)
  Name/number(s) of the lattice(s)

- SDUM
  blank   to use the transformation from the lattic routine
  ROT#nn   to use a ROT-DEFI rotation/translation from input
  name   the same as above but identifying the roto-translation by the name
  assigned in the ROT-DEFI SDUM (any alphanumeric string you like)

Example

```
LATTICE          Reg: TARGR1 ▼      to Reg:  ▼        Step:
   Id: 1tra ▼       Lat: 1.0         to Lat: 1.0       Step: 1.0
```

```
*...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+...
LATTICE         6.00000  19.00000              101.0000   114.00
```

Region # 6 to 19 are the "placeholders" for the first set replicas. We assign to them lattice numbers from 101 to 114

```
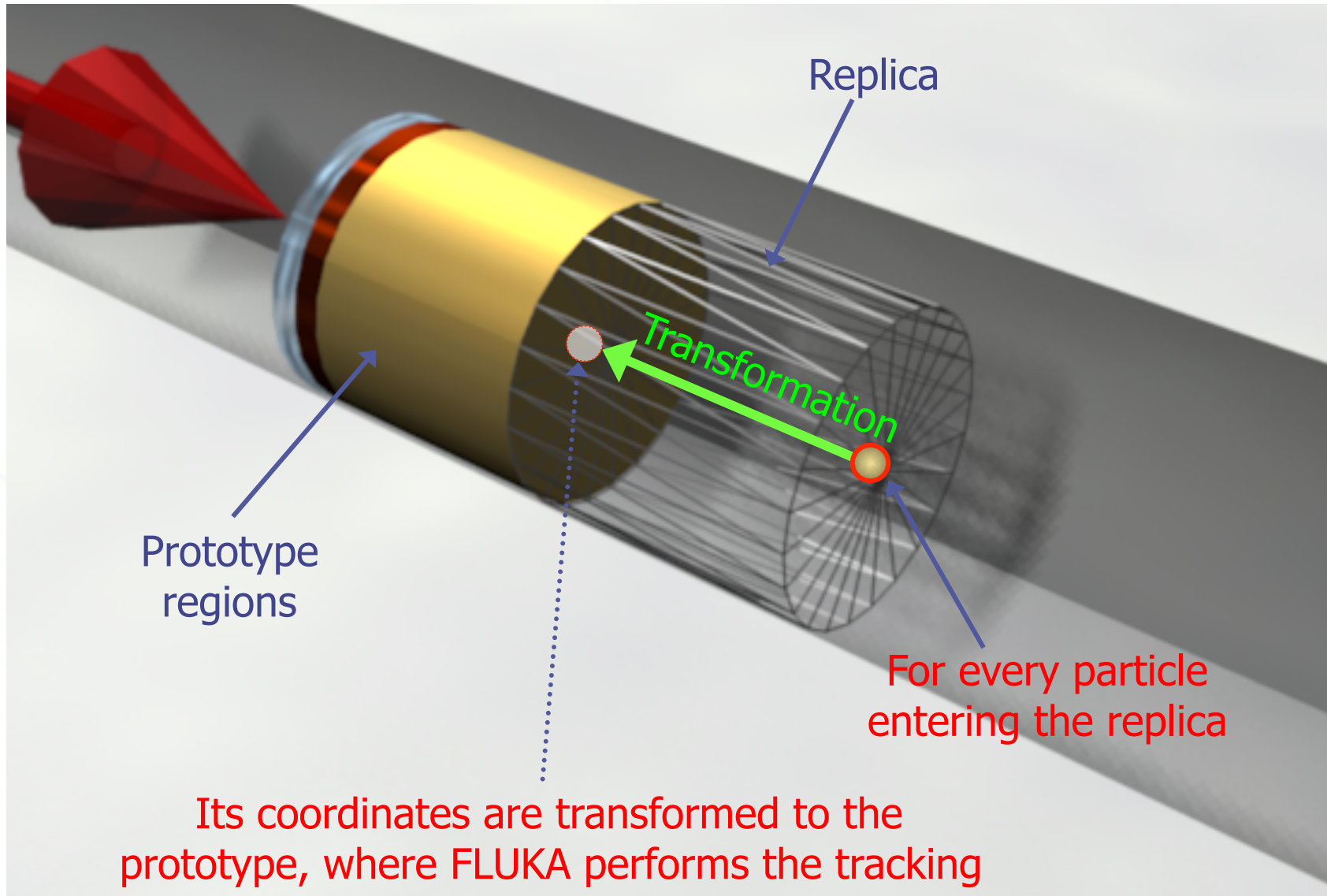LATTICE        TARGR1                          TargRep                    1tra
```

TARGR1 is the container region using transformation *1tra*

Example

Prototype regions

Empty lattice container

Example

Replica

Transformation

Prototype
regions

For every particle
entering the replica

Its coordinates are transformed to the
prototype, where FLUKA performs the tracking
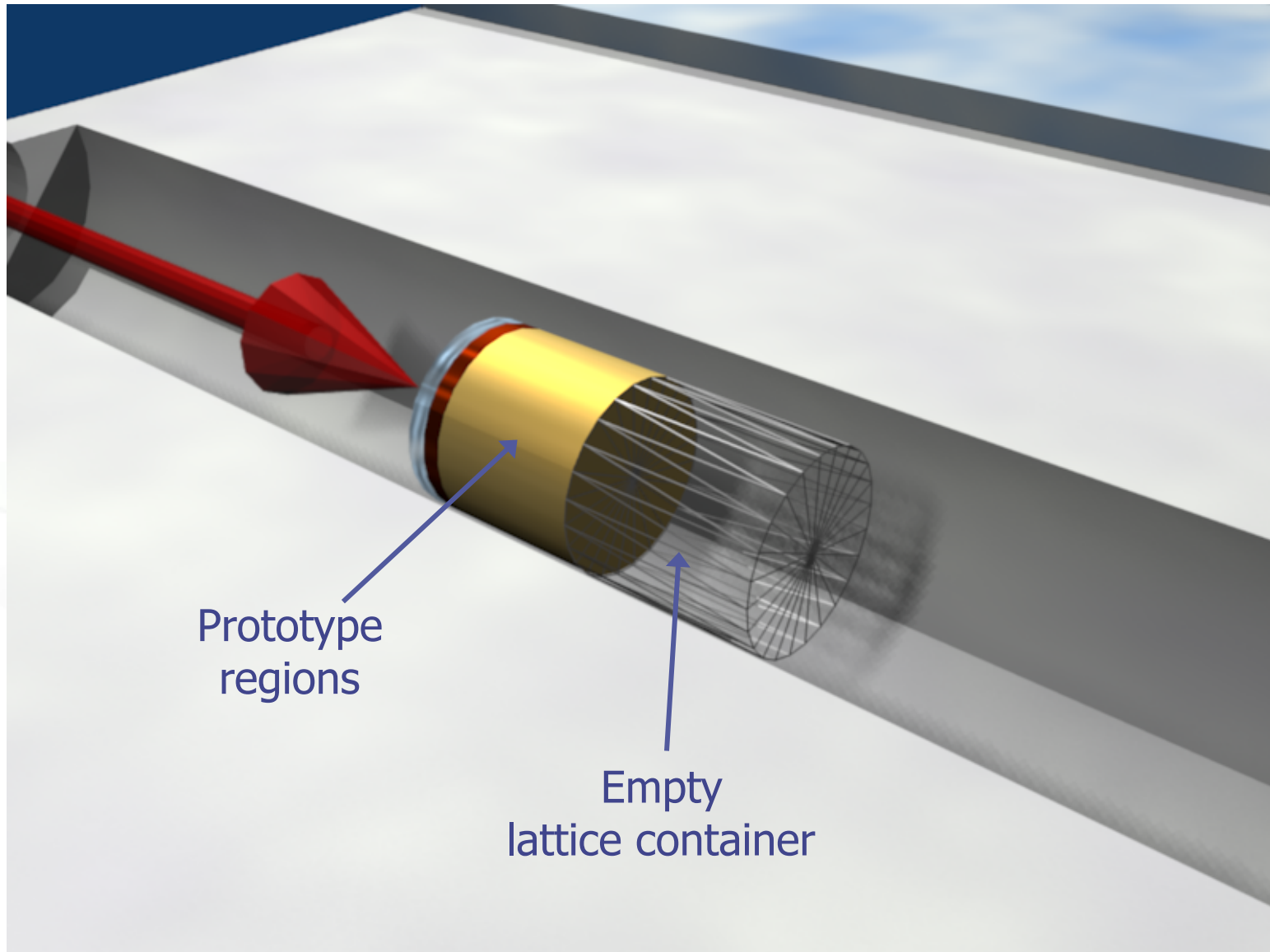
Example

Prototype cell

Empty lattice cell

Final replica

Transformation by input

- Rotations/Translations can be defined with the ROT-DEFIni card

- Can be assigned to a lattice by name or with ROT#nnn SDUM in the LATTICE card

- ROT-DEFIni cards can be consecutive (using the same index or name) to define complex transformations

WARNING:

Since matrix multiplication is not commutative the order of the Rotation/Translation operations in 3D is important.

Tips & Tricks


```
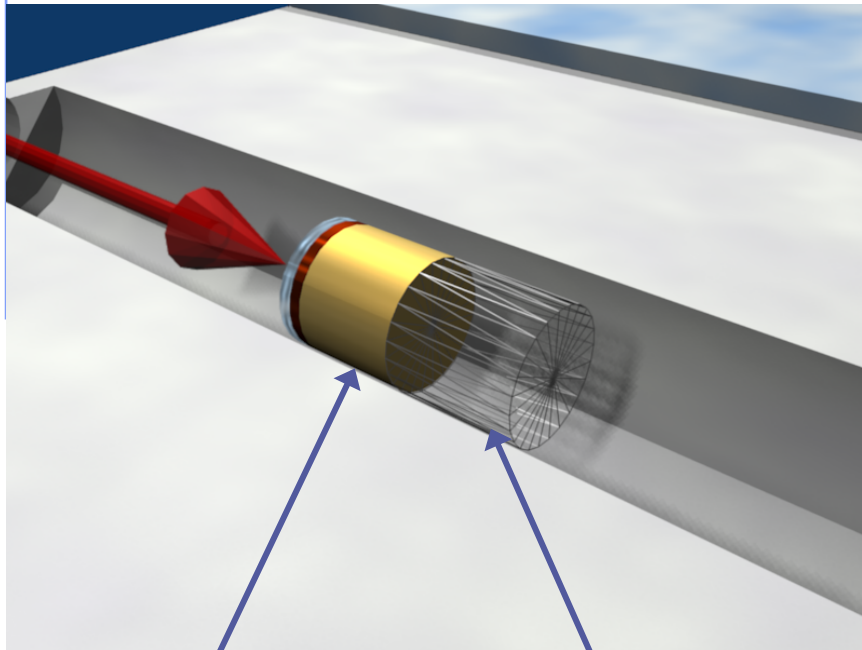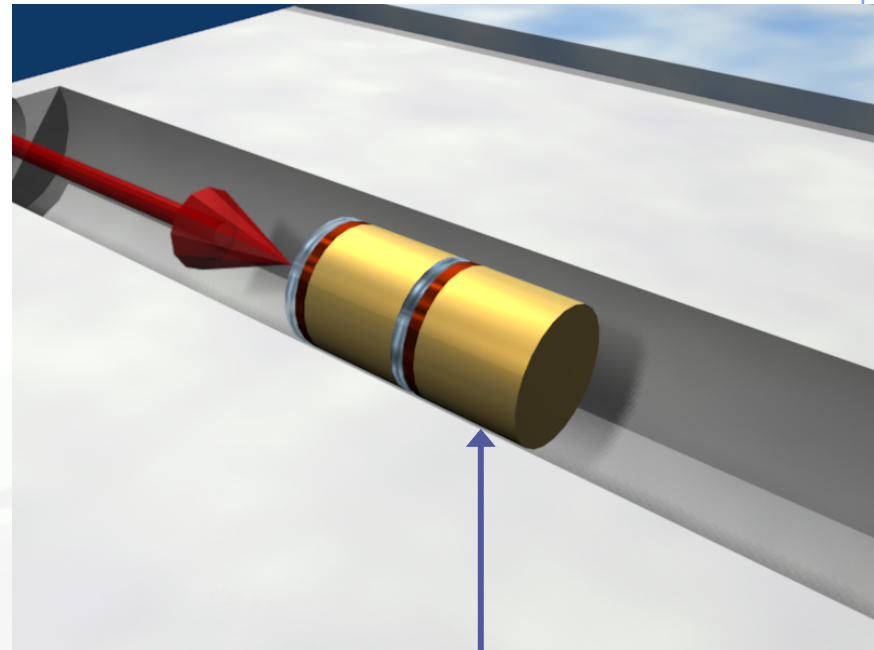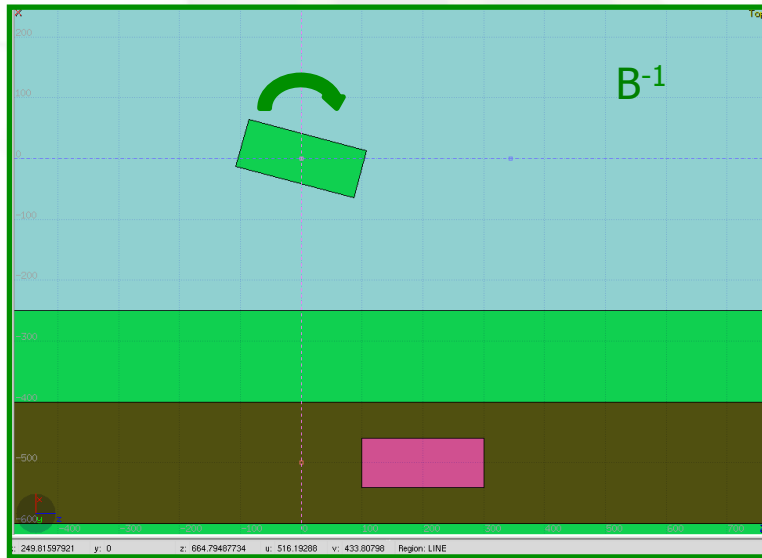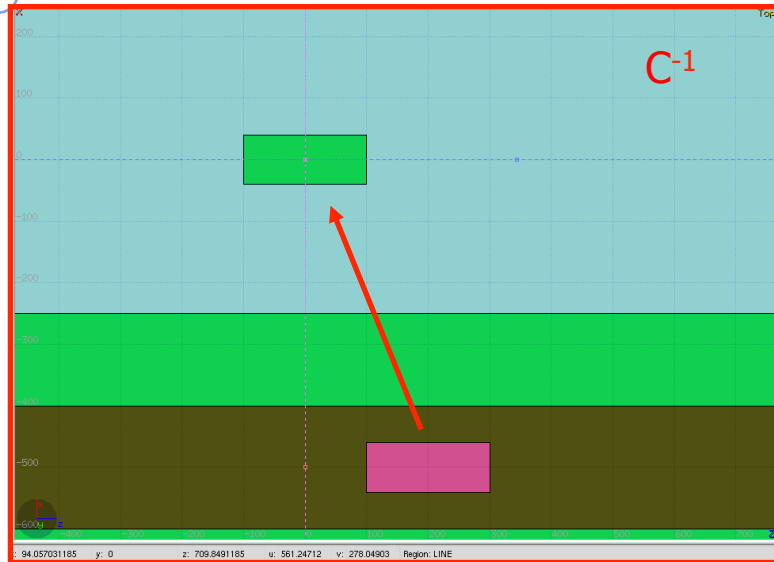GEOBEGIN
...
RPP CollProt -540.0 -460.0 -20.0 20.0 100.0 300.0
$start_transform –rotColl *
RPP CollRepl -540.0 -460.0 -20.0 20.0 100.0 300.0
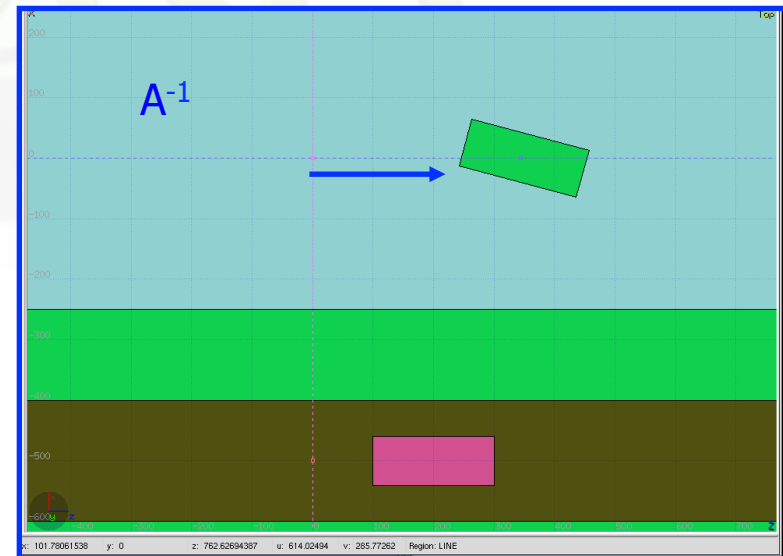$end_transform
...
REG 5  CollRepl
...
LATTICE, REG,,,CollRepl,,,rotColl
GEOEND
```

ROT-DEFI, 1.0, 0.0, 0.0, 0.0, 0.0, -350.0, rotColl   [A]

ROT-DEFI, 201.0, 0.0, -15.0, 0.0, 0.0, 0.0, rotColl   [B]

ROT-DEFI, 1.0, 0.0, 0.0, -500.0, 0.0, 200.0, rotColl [C]

* Remember: if R=CBA, then $R^{-1}=A^{-1}B^{-1}C^{-1}$

Tips & Tricks

- Always remember that the transformation must bring the container onto the prototype and not viceversa!

- You can always divide a transformation into many ROT-DEFI cards for easier manipulation.

- Rotations are always around the origin of the geometry, and not the center of the object.
  - To rotate an object, first translate the object to the origin of the axes
  - Perform the rotation
  - Move it by a final translation to the requested position.
    Of course with the inverse order since everything should apply to the replica

- In order to define the replica body, you can clone the body enclosing the prototype (assigning it a new name!) and apply to it the $Start_transform directive with the inverse of the respective ROT-DEFI transformation.

Tips & Tricks

- The Geometry transformation editor in flair can read and write ROT-DEFI cards with the transformation requested

- An easy way of creating a replica and the associated transformation is the following:

  1. Select the body defining the outer cell of the prototype
  2. Clone it with (Ctrl-D) and change the name of the clones. Click on "No" when you are prompted to change all references to the original name.
  3. Open the Geometry transformation dialog (Ctrl-T)
  4. Enter the transformation of the object in the listbox
  5. Click on "Transform" to perform the transformation on the clone bodies
  6. Click on "Invert" button to invert the order of the transformation
  7. Enter a name on the "ROT-DEFIni" field and click "Add to Input" to create the ROT-DEFIni cards
  8. Now you have to create manually the needed regions and the LATTICE cards

Numerical Precision

- Due to the nature of the floating point operations in CPU, even if the transformation looks correct the end result could be problematic
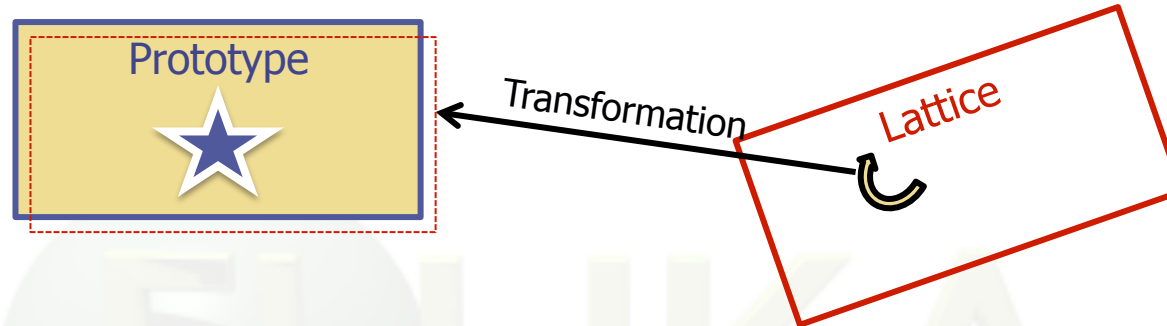
  This small misalignment between lattice/transformation/prototype could lead to geometry errors

- Use as many digits as possible to describe correctly the prototype and lattice cells as well as the transformation.
  It is mandatory that the transformation applied to the container makes the latter EXACTLY corresponding to the prototype

- One can use a FREE and FIXED card before and after the ROT-DEFI to input more than 9 digits

- GEOBEGIN WHAT(2) allows to relax the accuracy in boundary identification (USE WITH CAUTION)

Scoring with lattice

- **Materials and other properties** have to be assigned only to the regions constituting the prototype.

- In all (user) routines the **region number** refers to the corresponding one in the prototype.

- The SCORE summary in the .out file and the scoring by regions add together the contributions of the prototype region as well as of all its replicas!

- The lattice identity can be recovered runtime by the *lattice number*, as set in the LATTICE card, or available through the GEON2L routine if is defined by name

- In particular, the LUSRBL user routine allows to manage the scoring on lattices in the special USRBIN/EVENTBIN structure.

- USRBIN/EVENTBIN Cartesian and cylindrical meshes are geometry independent and, if superimposed on replica containers (possibly profiting from applying to the binning the respective roto-translation via ROTPRBIN), naturally discriminate between different replicas.

The USRBIN/EVENTBIN special binning

EVENTBIN or USRBIN with WHAT(1)=**8** :

Special user-defined 3D binning. Two variables are discontinuous (e.g. region number), the third one is continuous, but not necessarily a space coordinate.

Variable	Type	Default	Override Routine
1st	integer	region number	MUSRBR
2nd	integer	lattice cell number	LUSRBL
3rd	float	zero	FUSRBV

The lattic routine (I)

- The actual transformation from the lattice cell (container) to the elementary cell (prototype) can also be provided through the lattic routine (if the LATTICE SDUM is left blank)

  The use of the routine is mandatory for mirroring, and turns out to be highly preferable in the case of a lot of replicas placed according to a simple arithmetical rule (e.g. segmented detectors). Otherwise, the use of the LATTICE and ROT-DEFI cards does not imply any particular limitation and offers the possibility of being combined with the $Start_transform directive for the container definition (see later) and with the **use of ROTPRBIN for the roto-translation of USRBIN scoring grids**.

  SUBROUTINE LATTIC (XB, WB, DIST, SB, UB, IR, IRLTGG, IRLT, IFLAG)
  - IRLTGG is the current lattice number (it can optionally be set in the LATTICE card), IR is the current region number
  - XB,WB are vectors with the current particle position and direction
  - the routine must give back SB,UB, i.e. position and direction *transported to the prototype*

  The

  ENTRY LATNOR (UN, IRLTNO, IRLT)

  must provide the transformation for a vector representing a direction (no translation), applying to boundary normals (UN is both the in and out vector; IRLTNO is the current lattice number)

The lattic routine (II)

As an example, for the *10cm translation along z* shown in the previous slides:

```
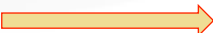LOGICAL LFIRST
DATA LFIRST / .TRUE. /
SAVE LFIRST, IREP
IF (LFIRST) THEN  ! Find replica's lattice number
        CALL GEON2L('TargRep ', IREP, IRTLAT, IERR)
        LFIRST = .FALSE.
END IF
IF ( IRLTGG .EQ. IREP ) THEN
        SB (1) = XB (1)
        SB (2) = XB (2)
        SB (3) = XB (3) – 10.0D0
        UB (1) = WB (1)
        UB (2) = WB (2)
        UB (3) = WB (3)
END IF
```

And the UN transformation is the identity

More complex cases can involve reflections and rotations.
For instance, for a reflection around the z axis :
```
UN (1) = UN (1)
UN (2) = UN (2)
UN (3) =-UN (3)
```

Accessing ancillary core routines

- To convert the lattice/region name of interest into the respective number (and vice versa), use the following routines – giving back IERR=0 in case of success - :

  CHARACTER*8 LATNAM

  CALL **GEON2L**(LATNAM, NLATT, IRTLAT, IERR)   Lattice Name to Lattice #
  CALL **GEOL2N**(NLATT, LATNAM, IRTLAT, IERR)   Lattice # to Lattice Name

      IRTLAT is the returned index of the (possible) roto-translation associated

  CHARACTER*8 REGNAM

  CALL **GEON2R**(REGNAM, NREG, IERR)   Region Name to Region #
  CALL **GEOR2N**(NREG, REGNAM, IERR)   Region # to Region Name

- It is always a good practice to call them only the first time the calling user routine is accessed and save once for ever the needed pieces of info

59

Object runtime readjustment (I)

- how to implement collimator replicas set at different apertures?

LHC IR7
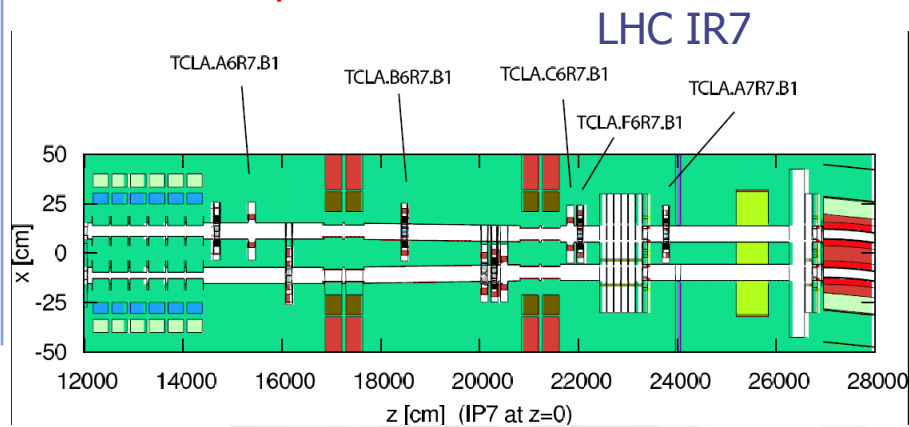
- how to implement the actual rock thickness according to the muon incident direction?

$$R^2 = d^2 + R_0^2 + 2R_0 d c_z$$

LNGS

muon

Atmosphere

Earth
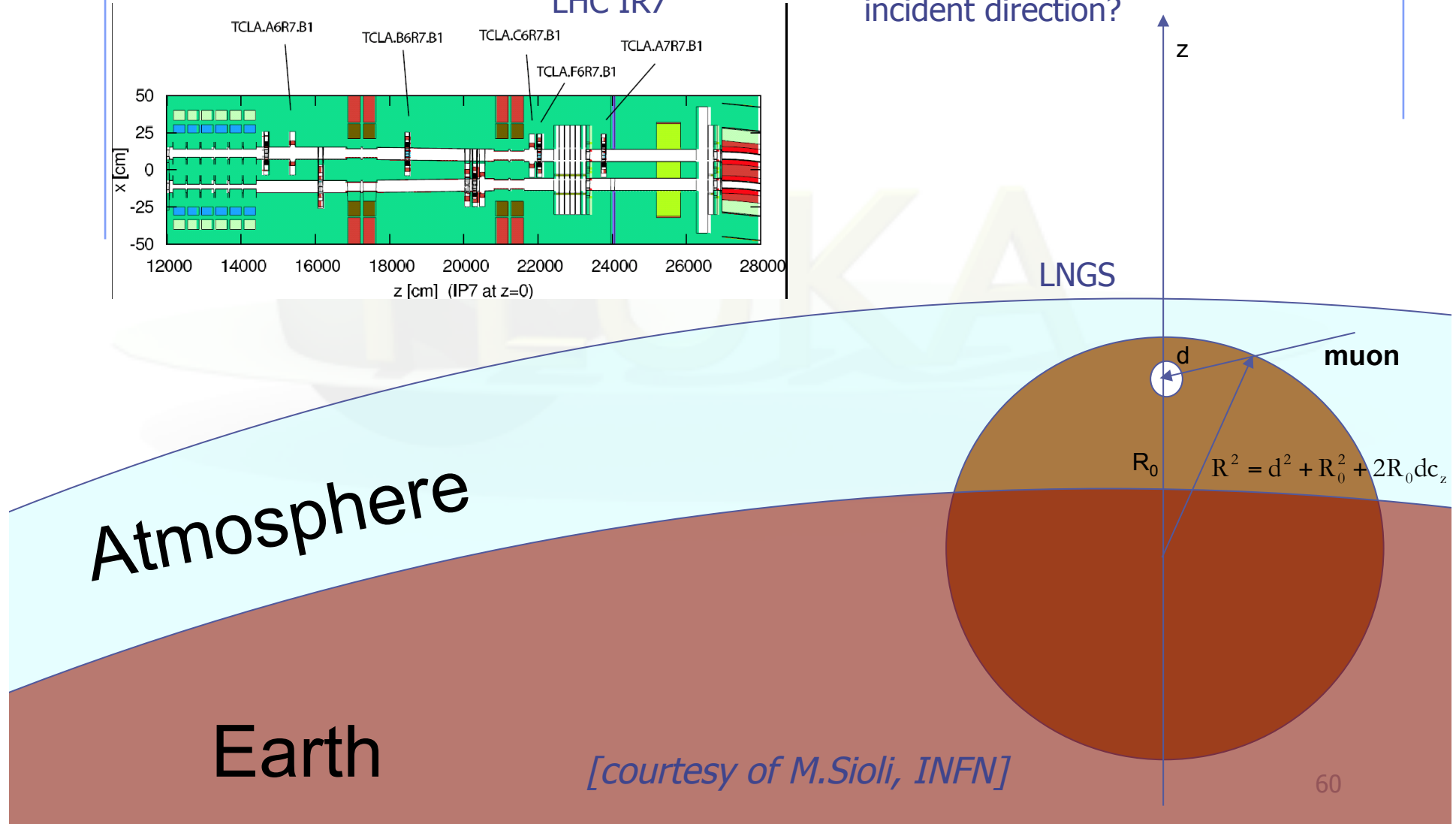
[courtesy of M.Sioli, INFN]

60

Object runtime readjustment (II)

In the lattic (when entering a container) and source (as well as usrmed) user routines, it is possible to manipulate the body parameters

CHARACTER*8 BODNAM
CALL **NM2BDY** (BODNAM, IBODY, IERR)   Body Name to Body #
DIMENSION BDYPAR(*NBDYPA*)
BDYPAR(I)=…   with I=1, NBDYPA
CALL **RSTBDY** (IBODY, ITYPE, BDYPAR, NBDYPA)   It forces recomputing distances only for IBODY

Body type	Type # [ITYPE]	# of parameters [NBDYPA]	Body type	Type # [ITYPE]	# of parameters [NBDYPA]
ARB	1	30	ZEC	11	4
SPH	2	4	XYP	12	1
RCC	3	7	XZP	13	1
REC	4	12	YZP	14	1
TRC	5	8	PLA	15	6
ELL	6	7	XCC	16	3
BOX	7	12	XEC	17	4
WED	8	12	YCC	18	3
RPP	9	6	YEC	19	4
ZCC	10	3	QUA	20	10

The FLUKA voxel geometry

- It is possible to describe a geometry in terms of "voxels", i.e., tiny parallelepipeds (all of equal size) forming a 3-dimensional grid
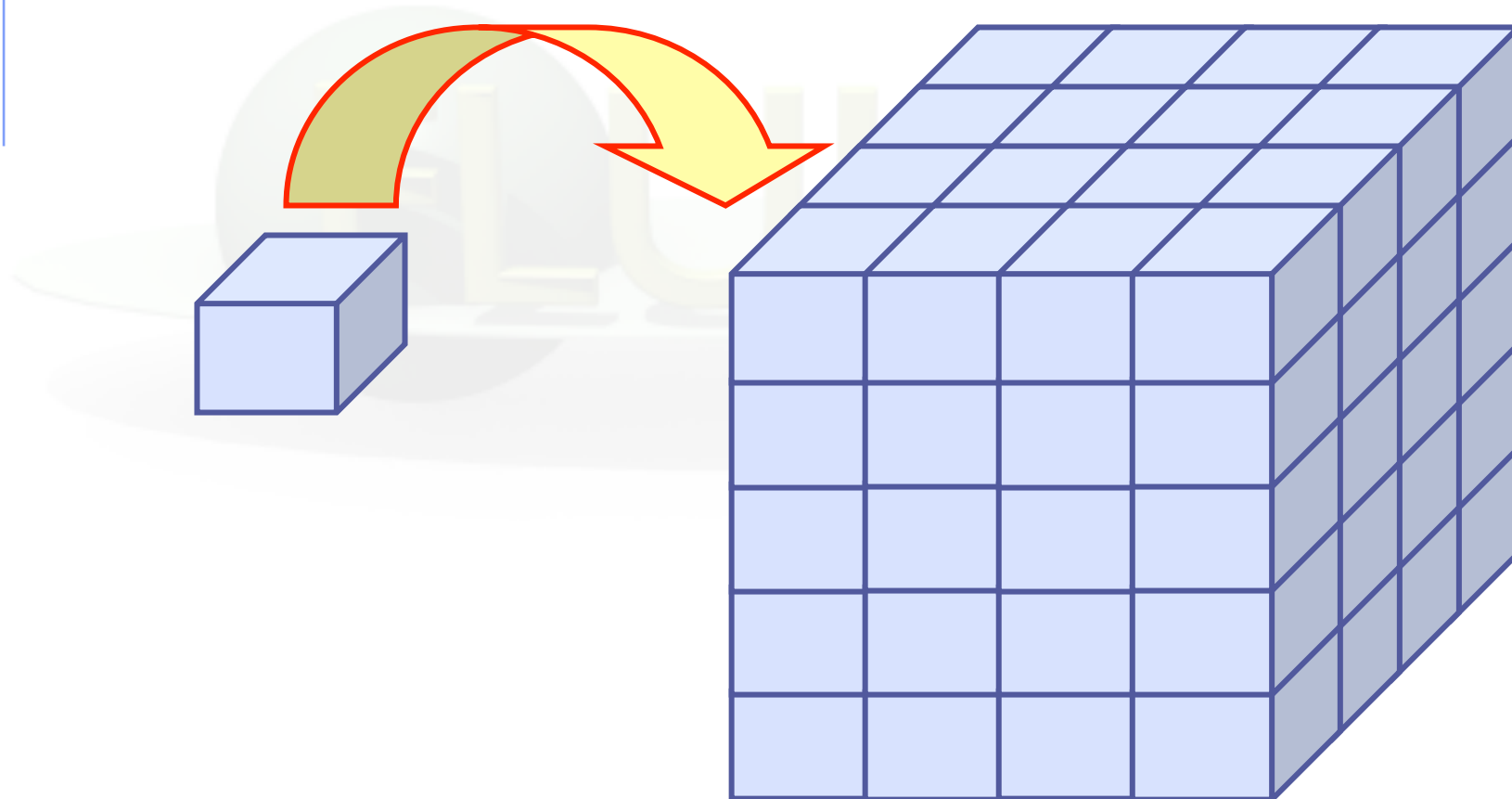
Voxel geometries: examples

The Gran Sasso in FLUKA
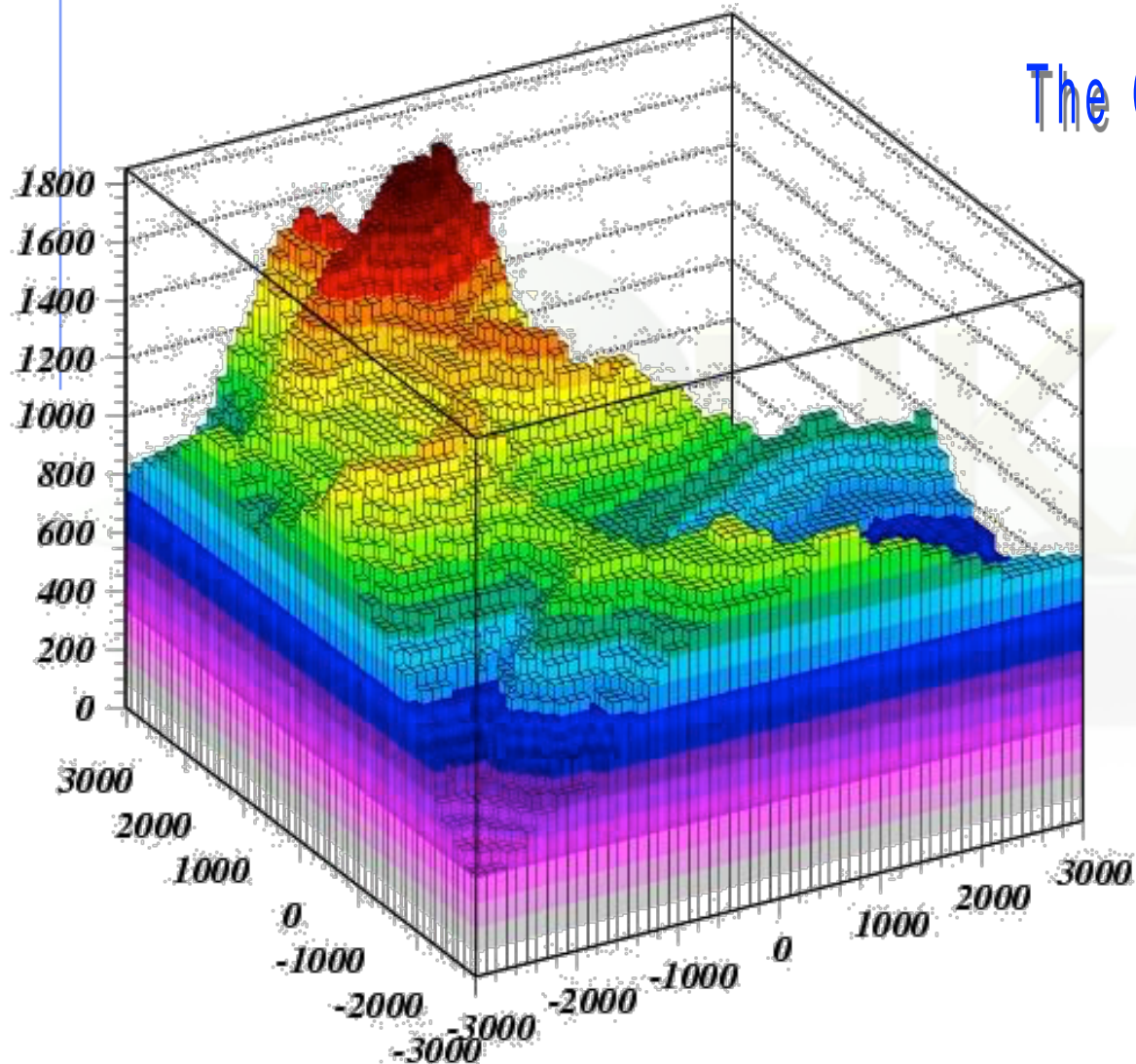
Voxel geometries in medical applications

- Voxel geometries are especially useful to import CT scan of a human body, e.g., for dosimetric calculations of the planned treatment in radiotherapy
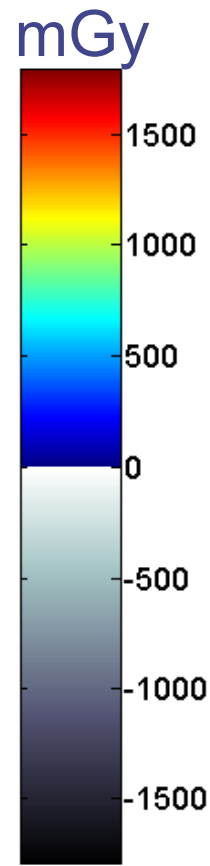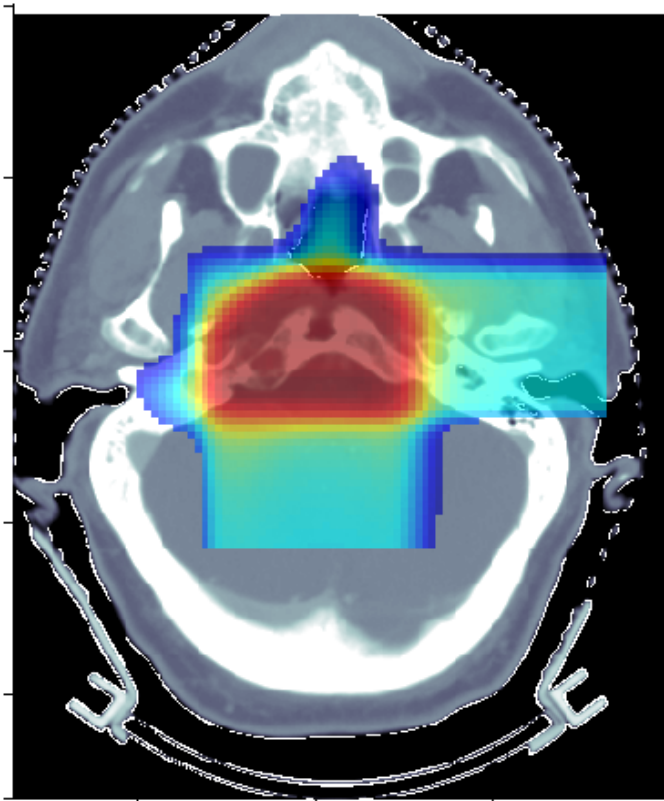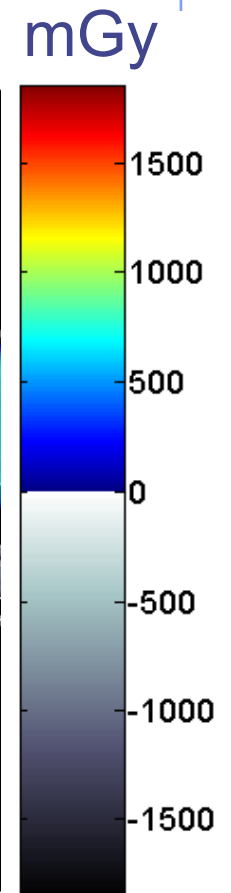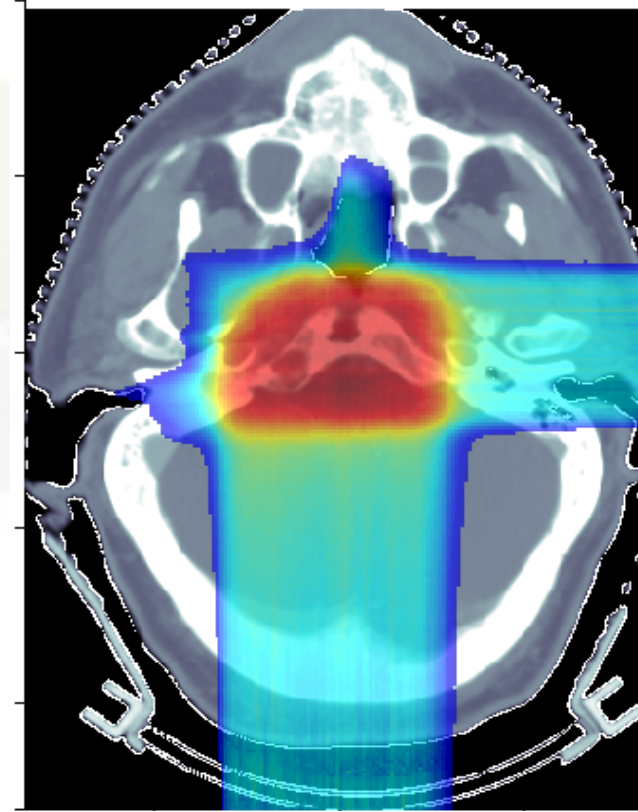
Parodi et al., 2007

Summary of relevant input cards

$Start_expansion $End_expansion

$Start_translat $End_translat   (inside the geometry input)

$Start_transform $End_transform

       to manipulate bodies

ROT-DEFI

       to define roto-translations

LATTICE   (inside the geometry input)

       to declare a region as a replica placeholder and associate it to a given transformation

VOXEL   (inside the geometry input)

       to introduce a voxel geometry