



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*

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)

Input format

- The input format for the geometry is different from that adopted elsewhere in FLUKA, i.e. the number and length of the input fields is different.
- The **recommended** format is **name based**. For backward compatibility there are also other formats. **Name based format is not the default one!**

Name based format input is used for both body and region if requested by **COMBNAME** in the **GEOBEGIN** card, or by a **GLOBAL** command at the beginning of the input file.

One advantage of name based format is that alignment of the input parameters is not necessary. Bodies and regions are identified by **names**. Its main advantages, in addition to the freedom from strict alignment rules, are the possibility to modify the input sequence without affecting the **region** description (for instance, by **inserting a new body**) and the availability of **parentheses to perform complex boolean operations** in the description of regions.

- In **fixed format** alignment is mandatory. Bodies and regions are identified by **numbers** and not by names which makes creation and updating of the geometry difficult.

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

The meanings of the **WHAT** and **SDUM** parameters are:

WHAT(1) flag for switching off geometry error messages: **don't touch!!**

Default = 0.0 (all geometry error messages are printed)

WHAT(2) used to set the **accuracy parameter** 

WHAT(3) = logical unit for the geometry input.

If > 0.0 and different from 5, the name of the corresponding file must be input on the next card. Otherwise, the geometry input follows.

WHAT(4) = logical unit for the geometry output. If > 0.0 and different from 11, the name of the corresponding file must be input on the next card. Otherwise, geometry output is printed on the 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 (!)**

Can be overwritten by **WHAT(5)** of a possible **GLOBAL** card

Tracking accuracy

$\text{WHAT}(2) * 10^{-6} \text{cm}$ is the *absolute accuracy (AA)* requested for tracking, applying to boundary identification.

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

So *AA* should be larger than $RA * L$, being L the largest coordinate value in the problem world (excluding the **outer blackhole shell** containing it), i.e. the whole geometry size.

From FLUKA2011, instead of giving $\text{WHAT}(2)$ the past default value of 0.0001, the code guesses L from the size of the second largest body in the geometry and sets $\text{WHAT}(2)$ 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
- a unique **body name**
(8 character maximum, alphanumeric identifier, case sensitive)
- a set of geometrical quantities defining the body
(their number depends on the body type)

A maximum of 132 characters per line are accepted,
use extra lines if required

The different items, separated by **one or more blanks**, or
by one of the separators **, / ; :**
can extend over as many lines as needed.

All numbers are in cm!

Generic quadric: QUA

A **QUA** is the most generic quadric surface

It is defined by 10 coefficients in the **following order**:

$$\mathbf{A}_{xx} \quad \mathbf{A}_{yy} \quad \mathbf{A}_{zz} \quad \mathbf{A}_{xy} \quad \mathbf{A}_{xz} \quad \mathbf{A}_{yz} \quad \mathbf{A}_x \quad \mathbf{A}_y \quad \mathbf{A}_z \quad \mathbf{A}_0$$

corresponding to the equation

$$\mathbf{A}_{xx} x^2 + \mathbf{A}_{yy} y^2 + \mathbf{A}_{zz} z^2 + \mathbf{A}_{xy} xy + \mathbf{A}_{xz} xz + \mathbf{A}_{yz} yz + \mathbf{A}_x x + \mathbf{A}_y y + \mathbf{A}_z z + \mathbf{A}_0 = 0$$

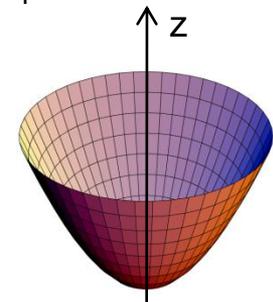
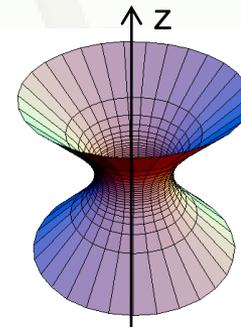
or $[x \quad y \quad z \quad 1]$ $\begin{bmatrix} \mathbf{A}_{xx} & \mathbf{A}_{xy}/2 & \mathbf{A}_{xz}/2 & \mathbf{A}_x/2 \\ \mathbf{A}_{xy}/2 & \mathbf{A}_{yy} & \mathbf{A}_{yz}/2 & \mathbf{A}_y/2 \\ \mathbf{A}_{xz}/2 & \mathbf{A}_{yz}/2 & \mathbf{A}_{zz} & \mathbf{A}_z/2 \\ \mathbf{A}_x/2 & \mathbf{A}_y/2 & \mathbf{A}_z/2 & \mathbf{A}_0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = 0$ i.e. $\mathbf{r}^T \mathbf{M}_{\text{QUA}} \mathbf{r} = 0$

For example:

QUA EllHyper 0.25 1.0 -4.0 0.0 0.0 0.0 0.0 0.0 0.0 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 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 0.0 -1.0 0.0
is an elliptic paraboloid with axis equal to z



Directives in geometry:expansion(&reduction)

➤ `$Start_expansion ... $End_expansion`

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

$$\mathbf{r}'^T M_{\text{QUA}} \mathbf{r}' = 0$$

$$\mathbf{r} = T \mathbf{r}'$$

$$T = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & f & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

`$Start_expansion 10.0`

SPH Sphere 5.0 7.0 8.0 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)

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

$$\mathbf{r}'^T M_{\text{QUA}} \mathbf{r}' = 0$$

$$\mathbf{r} = \mathbf{T} \mathbf{r}'$$

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & S_x \\ 0 & 1 & 0 & S_y \\ 0 & 0 & 1 & S_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

`$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)

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

$$\mathbf{r}'^T M_{\text{QUA}} \mathbf{r}' = 0 \quad \mathbf{r} = \mathbf{T} \mathbf{r}' \quad \mathbf{T} = \begin{bmatrix} & & & S_x \\ & \mathbf{R} & & S_y \\ & & & S_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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**)

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

- note that also the **inverse** transformation can be used

\mathbf{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.

$$\mathbf{X}_{\text{new}} = \mathbf{M}_{\text{polar}} \times \mathbf{M}_{\text{az}} \times (\mathbf{X} + \mathbf{T})$$

WHAT(1): assigns a transformation index and the corresponding rotation axis
I + J * 100 or **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 ($0 \leq \vartheta \leq 180^\circ$ degrees)

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

Directives in geometry: warnings

- `$Start_expansion` and `$Start_translat` are applied when reading the geometry → no CPU penalty, `$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 -50.0 -70.0 -80.0
```

```
$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
```

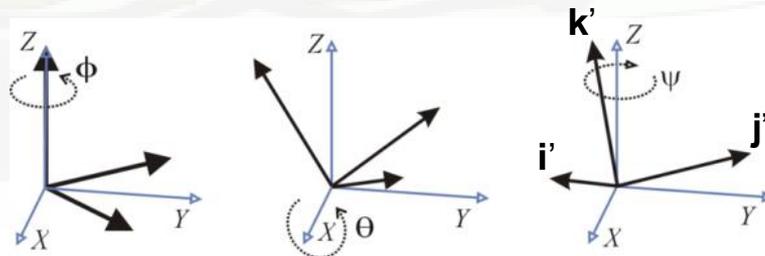
- Directives are not case sensitive (whereas roto-translation names are)

Identifying rotation angles

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

Then $[\mathbf{i}' \ \mathbf{j}' \ \mathbf{k}'] = \begin{bmatrix} c_1 c_3 - c_2 s_1 s_3 & -c_1 s_3 - c_3 c_2 s_1 & s_2 s_1 \\ c_2 c_1 s_3 + c_3 s_1 & c_1 c_2 c_3 - s_1 s_3 & -c_1 s_2 \\ s_3 s_2 & c_3 s_2 & c_2 \end{bmatrix}$ (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**)

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** is a rough guess for the number of zones which can be entered leaving the current region zones, it is 5 by default. What in fact matters is its **sum over all regions**, defining the size of the *contiguity list*.

At the beginning, to find the neighboring zones, the code searches over the whole geometry, but as the tracking proceeds, it learns the neighbors of each zone: they are stored in the contiguity list, making the calculation more and more efficient. When the above size limit is reached, the code prints a warning: GEOMETRY SEARCH ARRAY FULL. This is not lethal: the calculation continues but with a reduced efficiency. If the neighboring zones are not found in the contiguity list, the code will scan again ALL zones.

If you have more than 1000 regions, you must issue a **GLOBAL** card putting in **WHAT(1)** a higher limit (not beyond 20000)

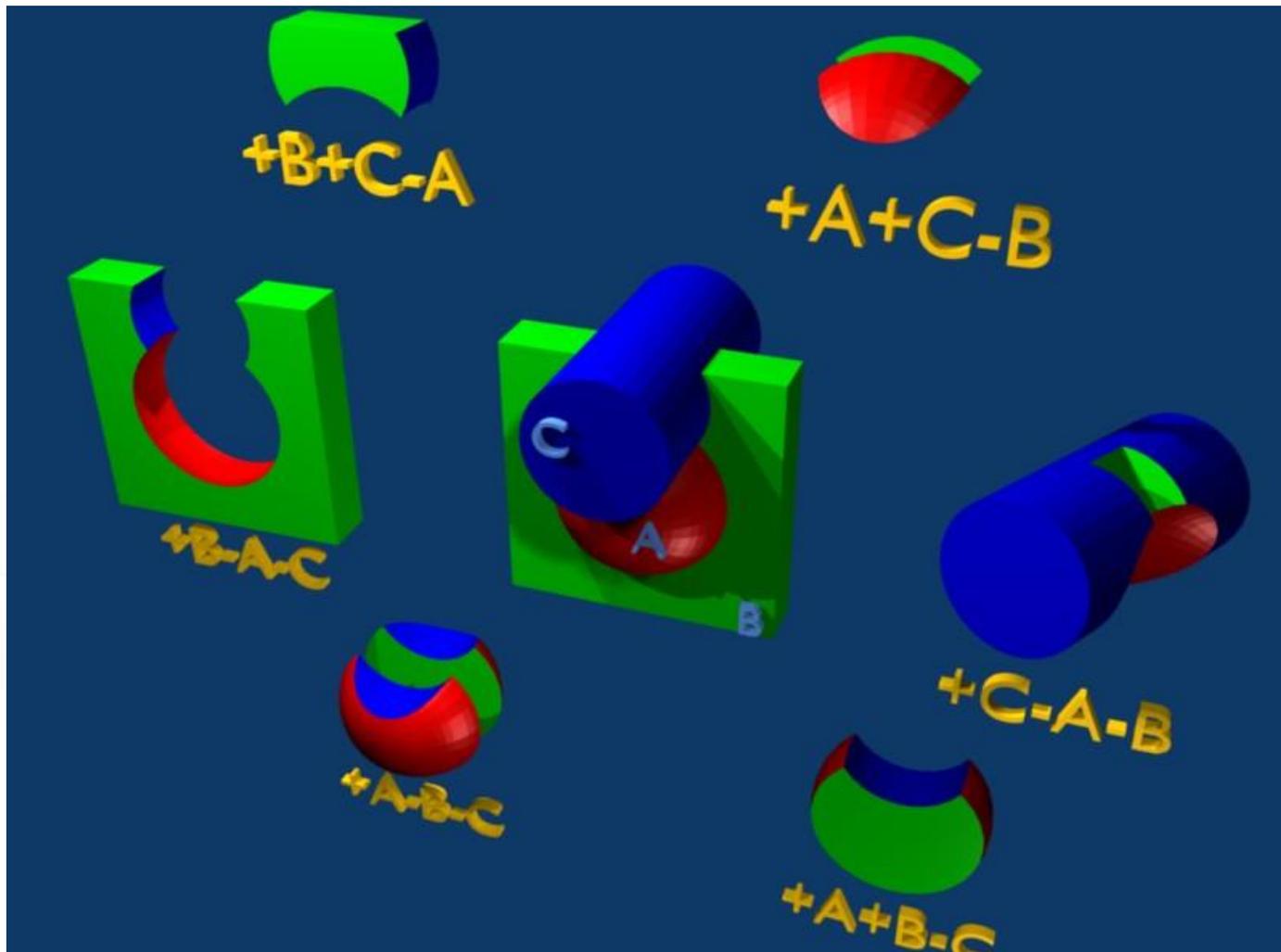
Operators

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

	Union	Subtraction	Intersection
Named based	 	-	+
<i>Fixed</i>	<i>OR</i>	-	+
Mathematically	\cup	-	\cap

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.

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:

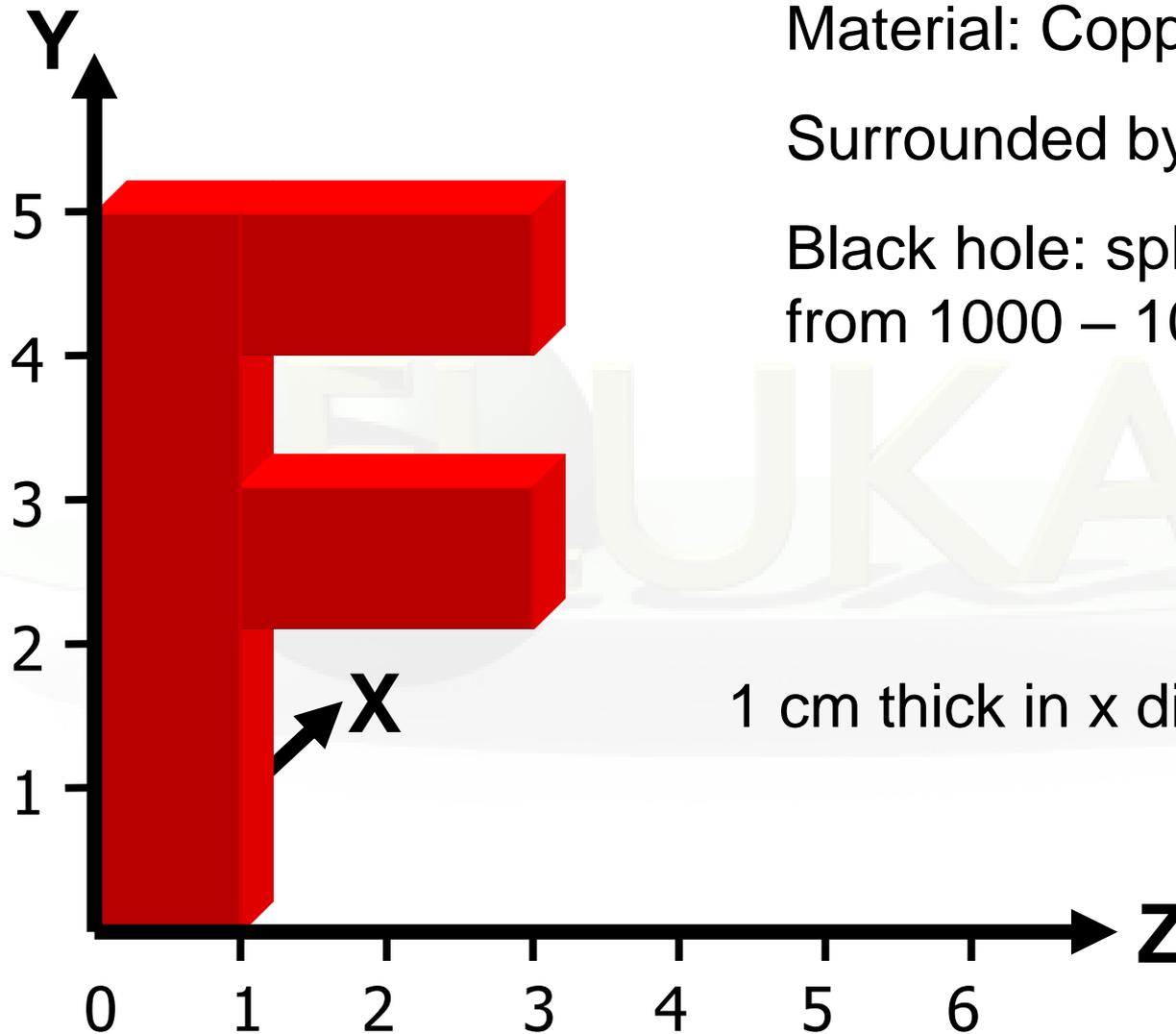
Each point of space must belong to one and only one region!

Geometry Example "F" shaped target

Material: Copper

Surrounded by vacuum

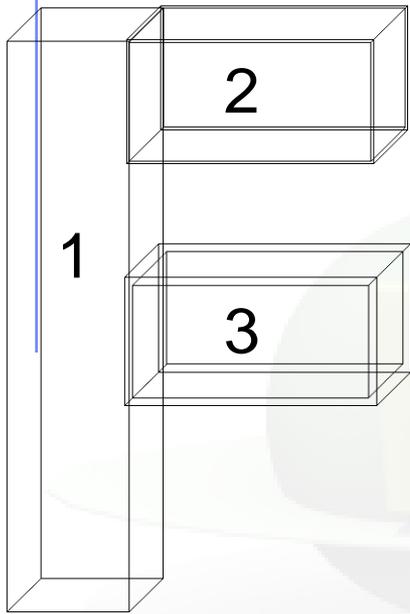
Black hole: spherical shell
from 1000 – 10000 cm



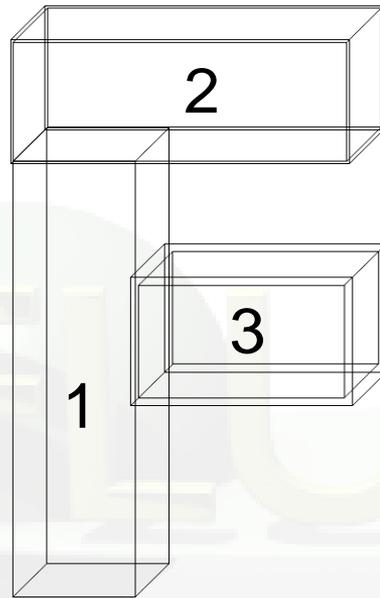
1 cm thick in x direction

Geometry example "F": Bodies

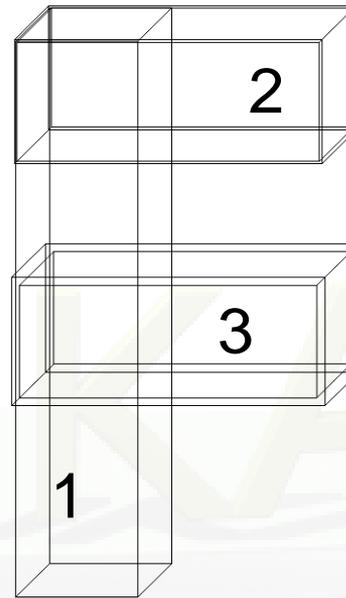
Several possibilities for bodies:



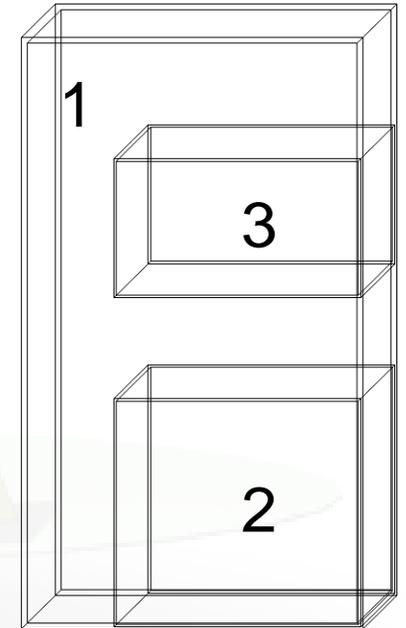
(A) 3 bodies



(B) 3 bodies



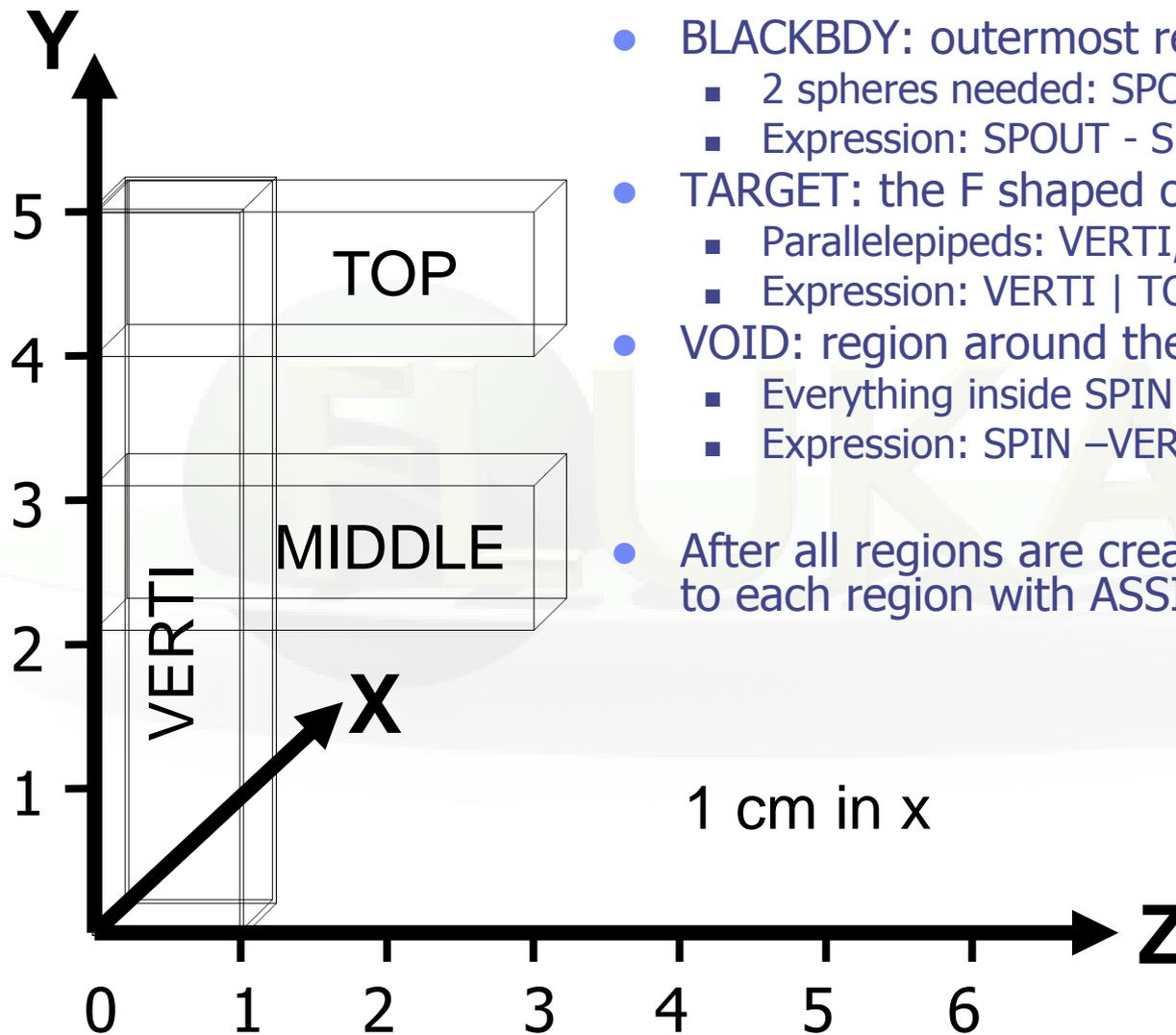
(C) overlapping



(D) subtraction

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

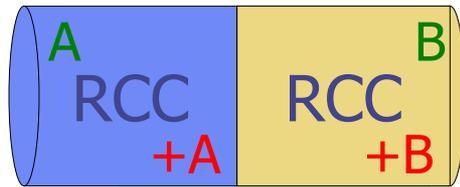
```
● GEOBEGIN COMBNAME
    0      0          The copper F
SPH SPIN      0.0 0.0 0.0 1000.
SPH SPOUT     0.0 0.0 0.0 10000.
RPP VERTI     0.0 1. 0.0 5. 0.0 1.
RPP TOP       0.0 1. 4. 5. 0.0 3.
RPP MIDDLE    0.0 1. 2. 3. 0.0 3.
END
* Black hole
BLKBODY      5  +SPOUT -SPIN
* Void around
VOID         5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

Flair view of the input file

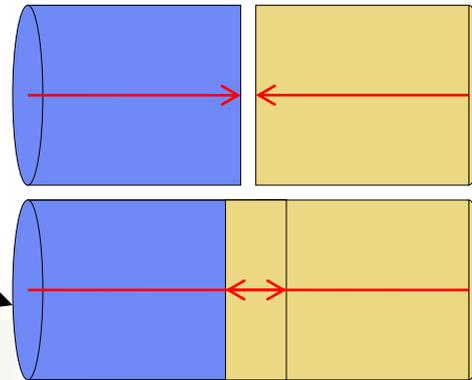
GEOBEGIN		Log: ▼	Acc:	Opt: ▼
Title: The copper F		Inp: ▼	Out: ▼	Fmt: COMBNAME ▼
SPH	SPOUT	x: 0 R: 10000	y: 0	z: 0
SPH	SPIN	x: 0 R: 1000	y: 0	z: 0
RPP	VERTI	Xmin: 0 Ymin: 0 Zmin: 0	Xmax: 1 Ymax: 5 Zmax: 1	
RPP	TOP	Xmin: 0 Ymin: 4 Zmin: 0	Xmax: 1 Ymax: 5 Zmax: 3	
RPP	MIDDLE	Xmin: 0 Ymin: 2 Zmin: 0	Xmax: 1 Ymax: 3 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: ▼
ASSIGNMA		Mat: VACUUM ▼	Reg: VOID ▼	to Reg: ▼ Step: ▼
ASSIGNMA		Mat: COPPER ▼	Reg: TARGET ▼	to Reg: ▼ Step: ▼

Preventing precision errors

Ideal world



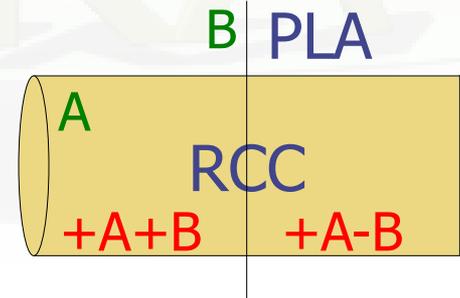
Computation world



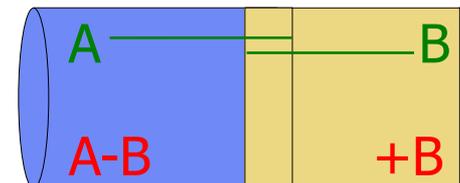
Avoid touching surfaces!

When floating point operations are involved

- Use cutting planes instead



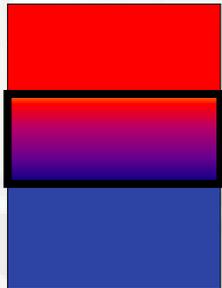
- Or force partial overlap of bodies



Geometry errors

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!!**



- **Primary particles starting on a boundary** (where the region is undefined) represent a **delicate** situation. FLUKA will try to determine the region by their direction.

Debugging tools

- **GEOEND** card with the **DEBUG** option
(*handled through a dedicated Flair frame*)
- 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				&

Native debugger (II)

- If **no error** is found, no **.err** file will be created.
- 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 as step length an irrational number in order to prevent from ending up on "special" points (i.e. boundaries)
- **REMINDER:** If the debugger doesn't find any error it doesn't mean that the geometry is error free!

Native debugger (III)

It can be easily run through Flair

The screenshot shows the Flair software interface with the Geometry Debugger window open. The main window displays a table of regions with 'Target Area' selected. A smaller 'Geometry ...' dialog box is overlaid on the right, showing details for the selected region.

Name	Region
Target Area	$[-45.0, -55.0, -40.0] - [45.0, 55.0, 40.0] \times (51, 51, 51)$
Region #22222	$[10.0, 20.0, 30.0] - [0.0, 0.0, 0.0] \times (1, 1, 201)$
Region #3aaa	$[22.0, 0.0, 0.0] - [0.0, 0.0, 0.0] \times (101, 10, 1)$
Region #4bbb	$[33.0, 0.0, 0.0] - [0.0, 0.0, 0.0] \times (1, 1, 1)$

Geometry ...

Region: Target Area

Min: [-45.0, -55.0, -40.0]

Max: [45.0, 55.0, 40.0]

Bins: [51, 51, 51]

Elapsed: 2 s

Status: Finished with no errors!

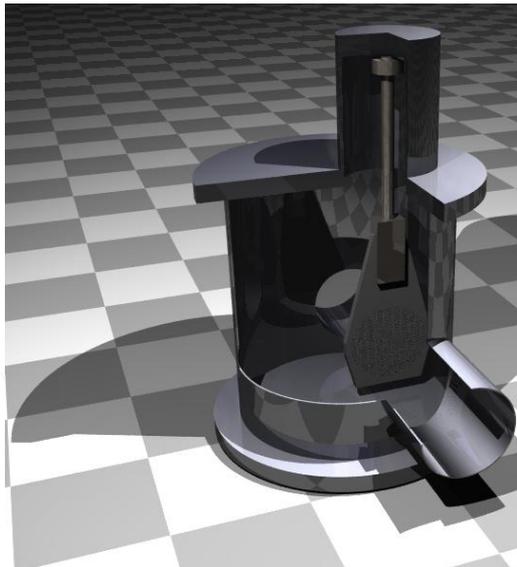
View Close

Debug

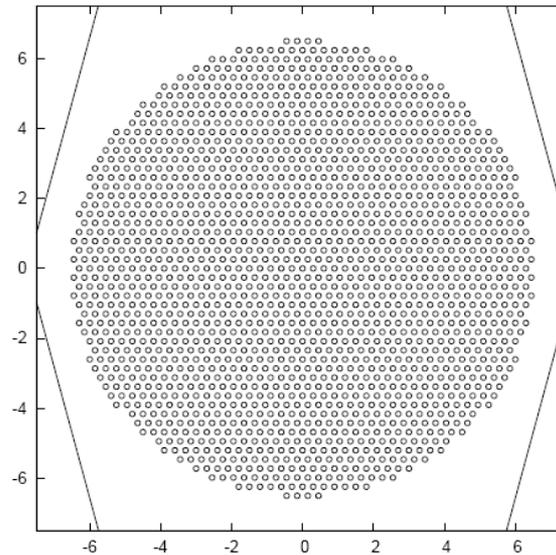
Inp: ntof33.inp Exe: Dir: /home/bnv/prg/physics/fluka/flair/examples

User optimization in region definition

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

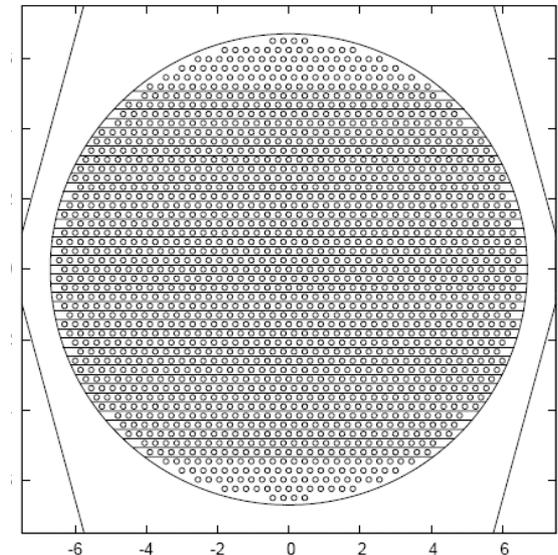


1731 holes



one zone with MANY bodies:
large CPU time!

slices with <50 holes



many zones with much less
bodies each: significant benefit

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



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:
 $A + A = A$
 $A - A = \emptyset$
 $\text{exp}A | \text{exp}A = \text{exp}A$

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 (I)

FLUKA geometry has *replication* (lattice) capabilities

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

[In a future release the possibility of a second level is planned]

- 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 (`latic.f`)

The lattice identification is available for scoring

Transformations are limited to:

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

WARNING:

Do not use scaling or any deformation of the coordinate system

Lattice (II)

- The regions which constitute the **elementary cell** (*prototype*) to be replicated, have to be defined in detail. **Materials and other properties** have to be assigned only to the **regions** constituting the **prototype**.
- 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
- **Non-replicas carry the lattice number 0**
- Lattices and plain regions can coexist in the same problem

LATTICE card

After the Region 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 *latic* 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...+...
```

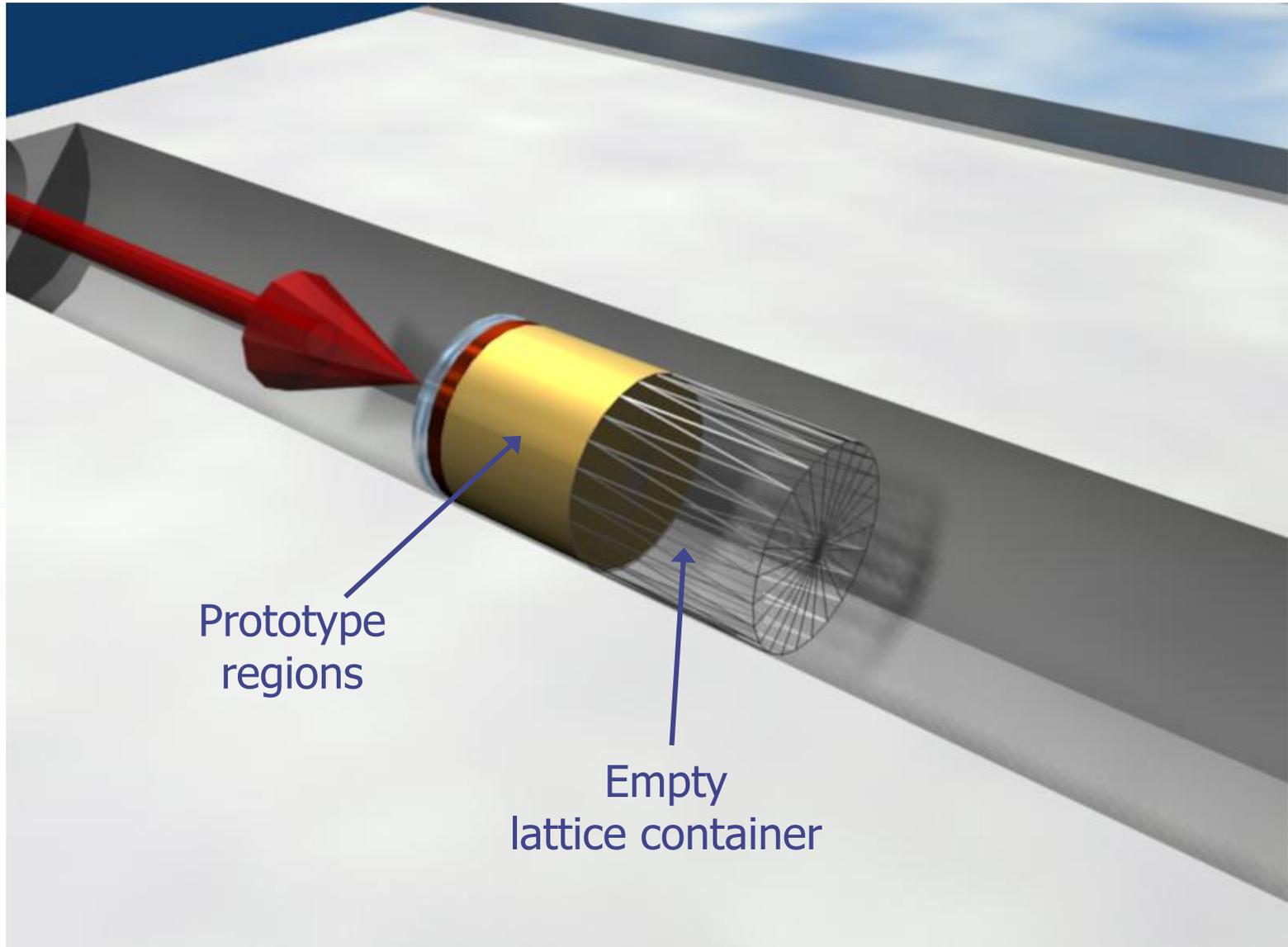
```
LATTICE          TARGR1          TargRep          1tra
```

TARGR1 is the container region using transformation *1tra*

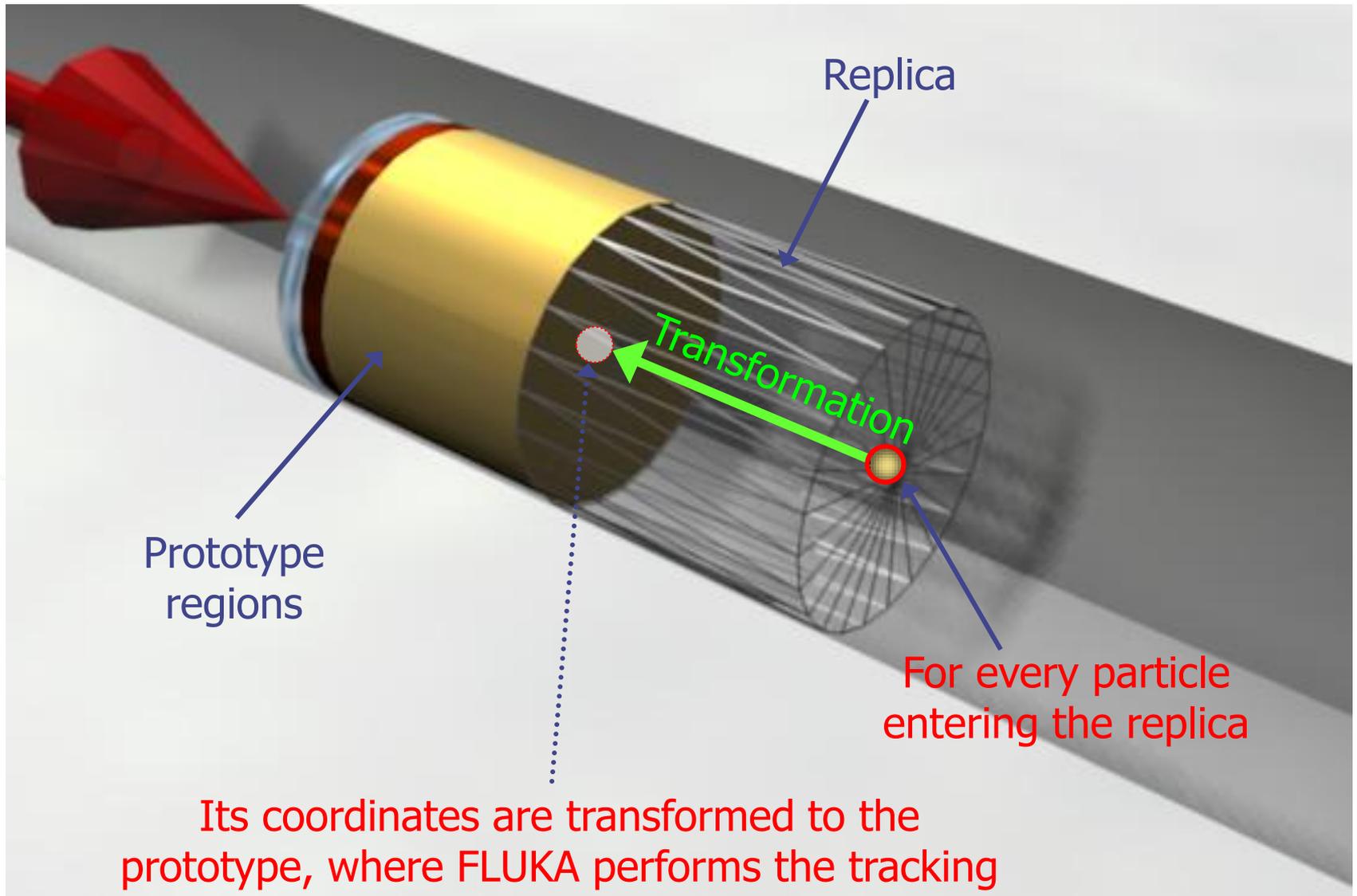
```
LATTICE          6.00000  19.00000          101.0000  114.00
```

Region # 6 to 19 are "placeholders", to which we assign lattice numbers from 101 to 114

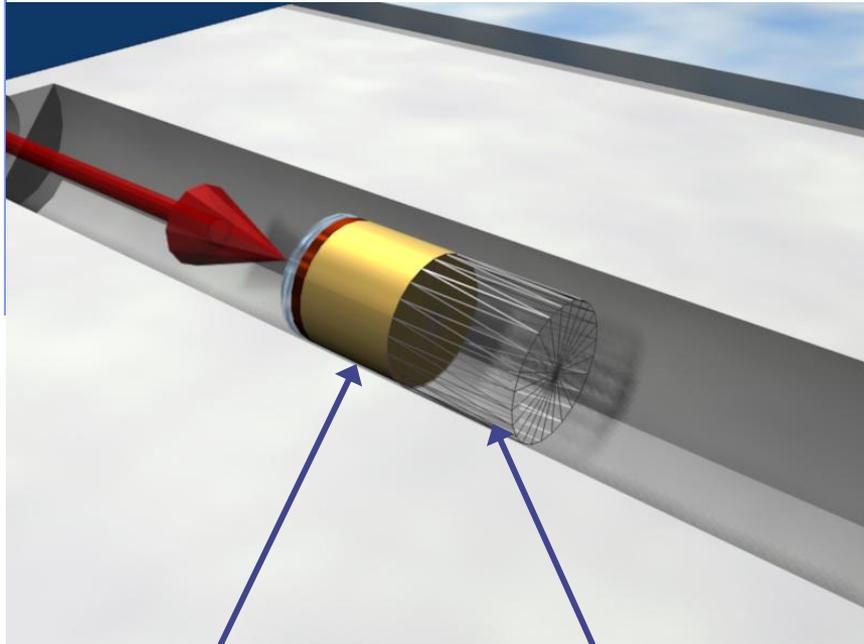
Example (I)



Example (II)

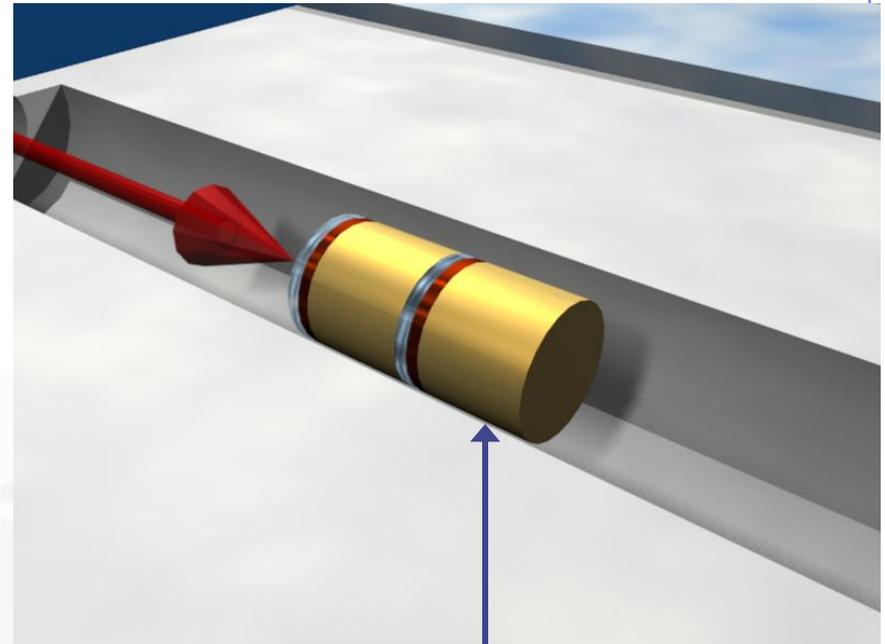


Example (III)



Prototype cell

Empty lattice cell



Final replica

Transformation by input card

- 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 concatenated (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.

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
```

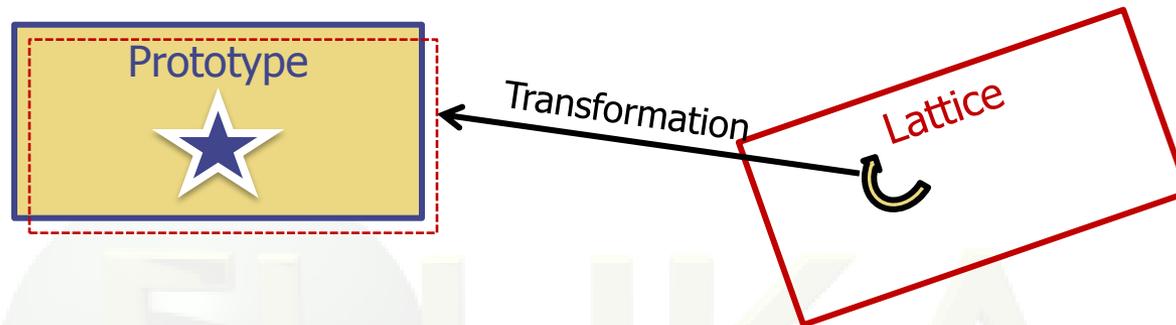
And the **UN** transformation is the identity

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)
```

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



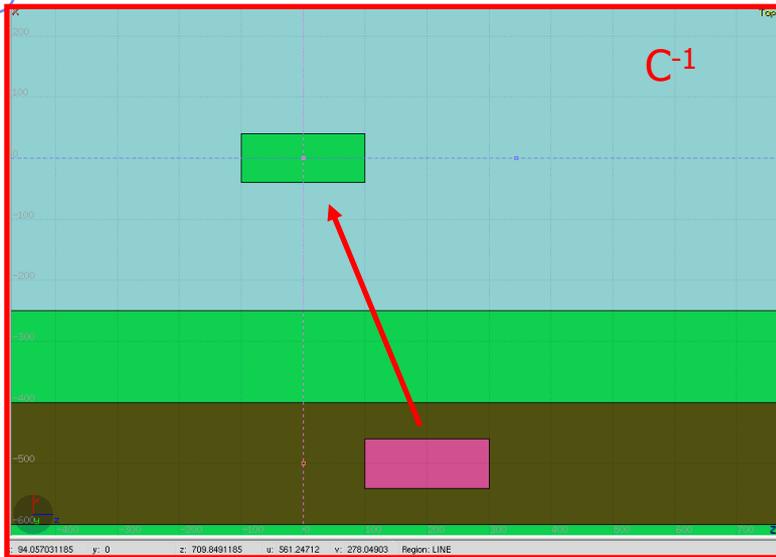
This small misalignment between lattice/transformation/prototype could lead to geometry errors

- 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** card to input more than 9 digits
- **GEOBEGIN WHAT(2)** allows to relax the accuracy in boundary identification (**USE WITH CAUTION**)

Rules and strategies

- 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 more clarity.
- 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.
Actually the other way around 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. 

A robust solution: combining lattice with body transformation



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

```
...
Collimat 5 +CollRepl
END
```

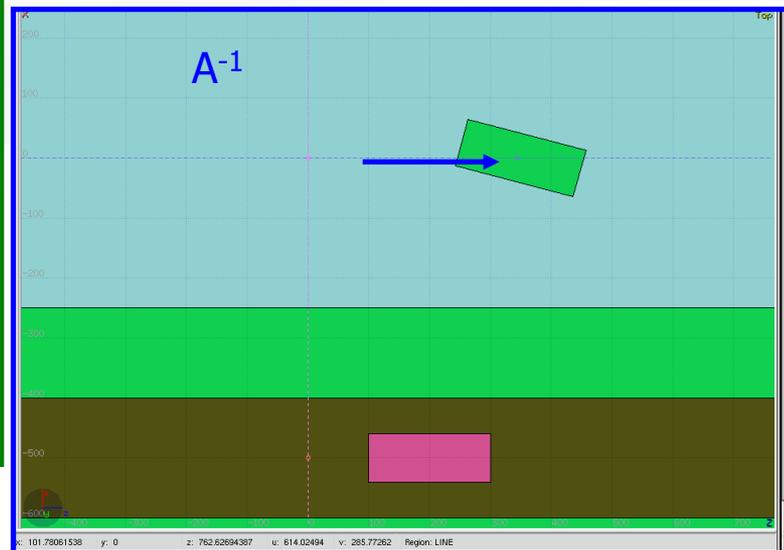
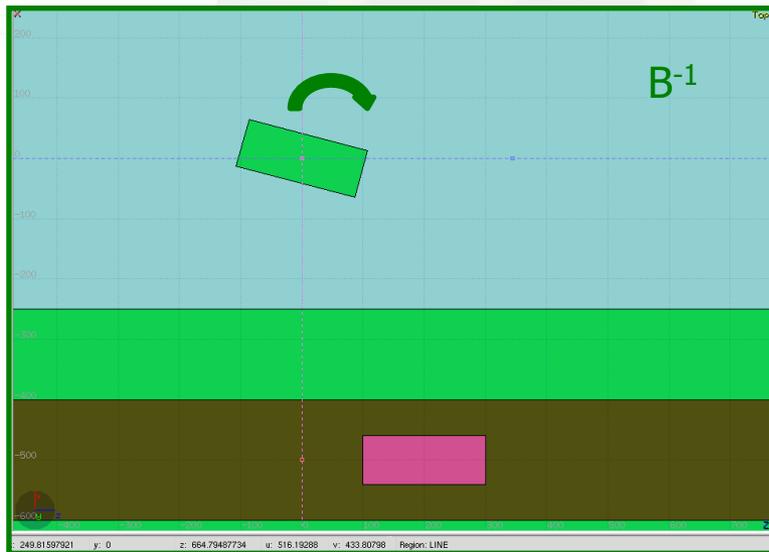
LATTICE, Collimat, , , Whatever, , , **rotColl**

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]



Scoring with lattice

- 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 from the lattice name
 - In particular, a **special** built-in **USRBIN/EVENTBIN** structure is available to manage the scoring on lattices through the **LUSRBL** user routine.
 - **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/USRBIN with WHAT(1)=**8** and **18** :

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

Variable	Type	Default	Override Routine
1 st	integer	region number	MUSRBR
2 nd	integer	lattice cell number	LUSRBL
3 rd	float	no default*	FUSRBV

* Presently it returns 0

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

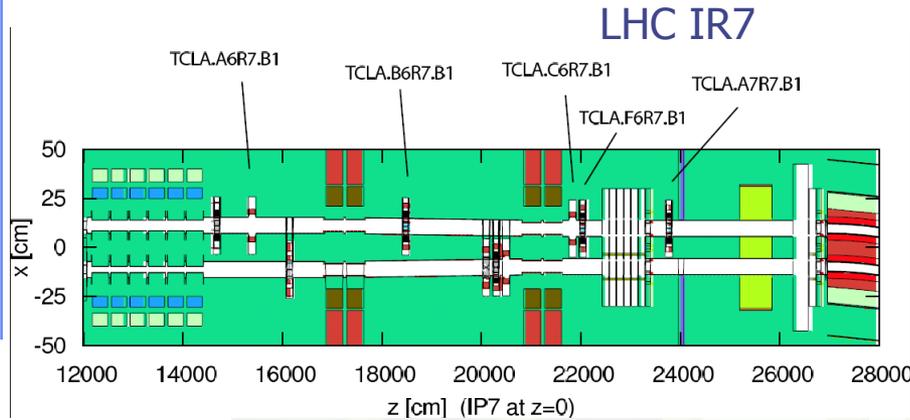
CALL **GEON2R**(REGNAM, **NREG**, IERR) Region Name to **Region #**

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

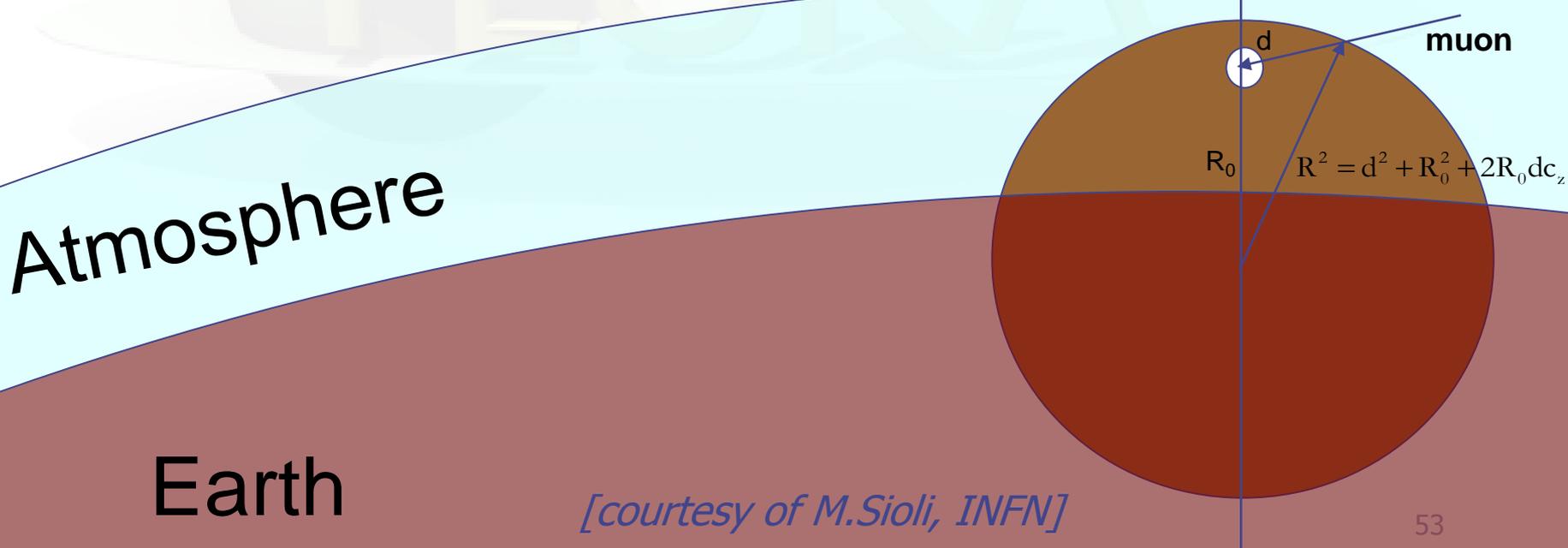
- It is always a good practice to **call them only the first time** the calling user routine is accessed and save once for ever the needed pieces of info

Object runtime readjustment (I)

- how to implement **collimator replicas** set at different apertures?



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



[courtesy of M.Sioli, INFN]

Object runtime readjustment (II)

In the **latic** (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
```

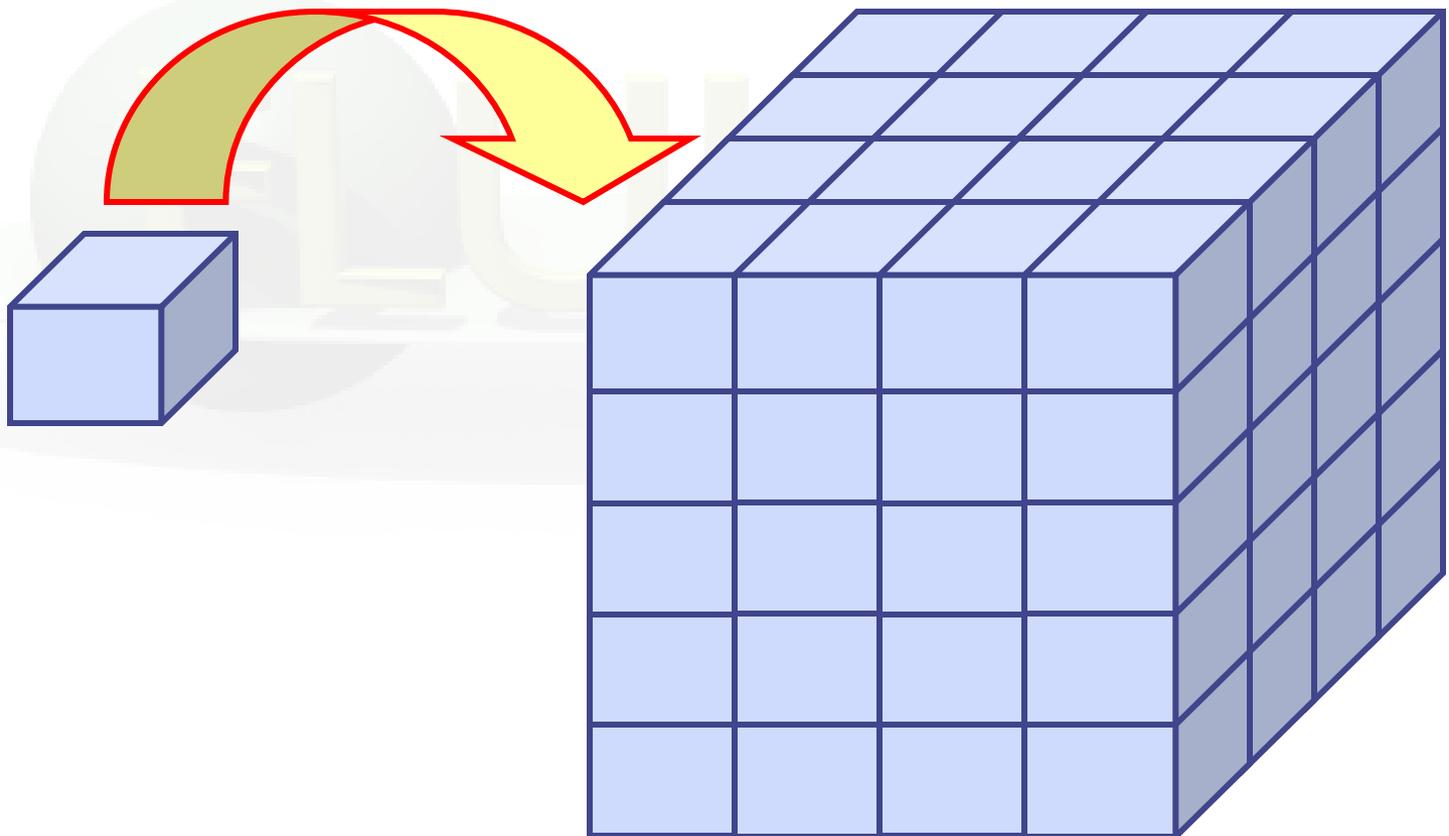
```
CALL RSTBDY (IBODY, ITYPE, BDYPAR, NBDYPA)
```

It forces recomputing distances only for **IBODY**

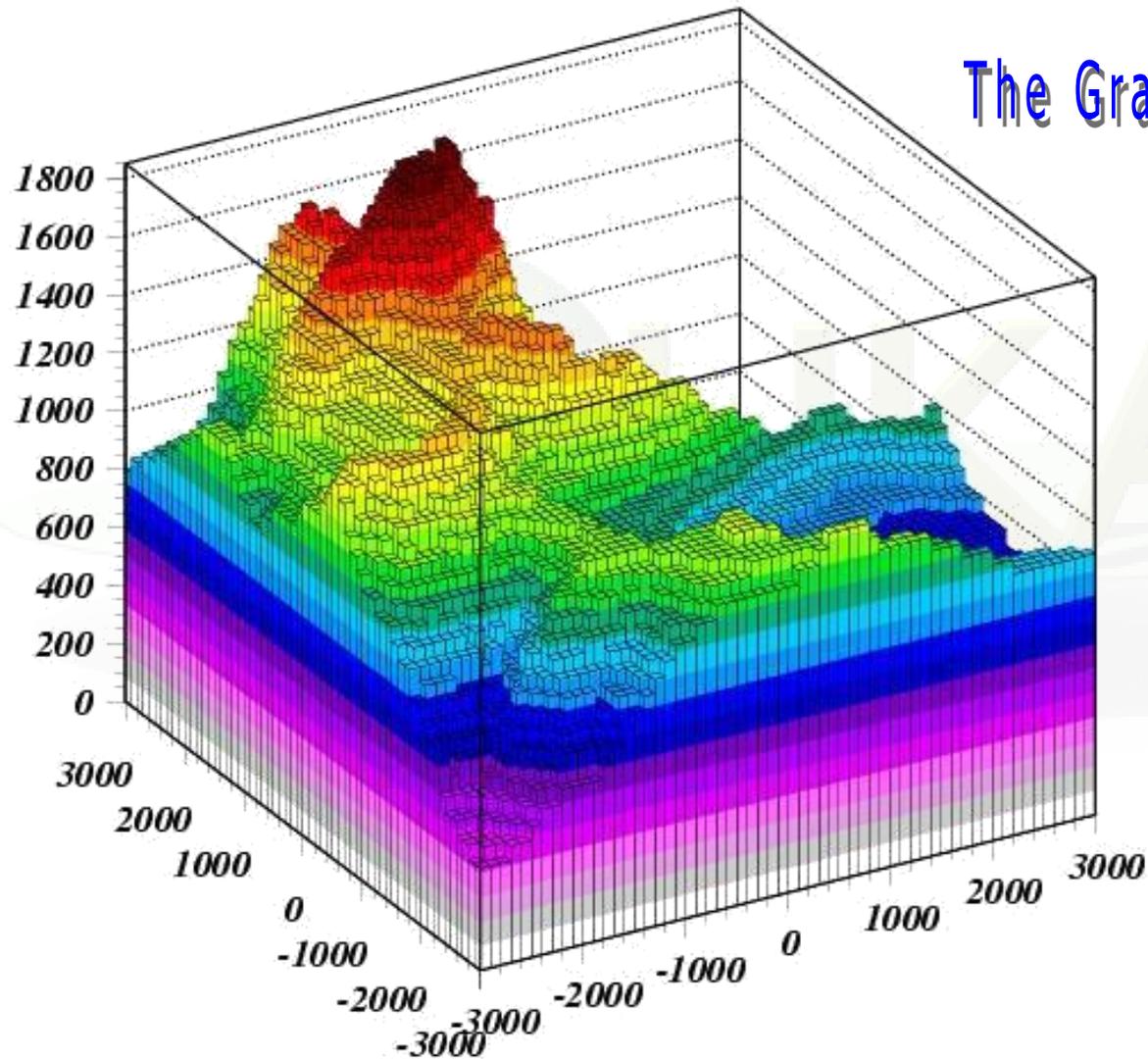
Body type	Type # [ITYPE]	# of parameters [NBDYPA]	Body type	Type # [ITYPE]	# of parameters [NBDYPA]
ARB	1	30	ZEC	11	4
SPH	2	4	XYP	12	1
RCC	3	7	XZP	13	1
REC	4	12	YZP	14	1
TRC	5	8	PLA	15	6
ELL	6	7	XCC	16	3
BOX	7	12	XEC	17	4
WED	8	12	YCC	18	3
RPP	9	6	YEC	19	4
ZCC	10	3	QUA	20	10

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**



An example

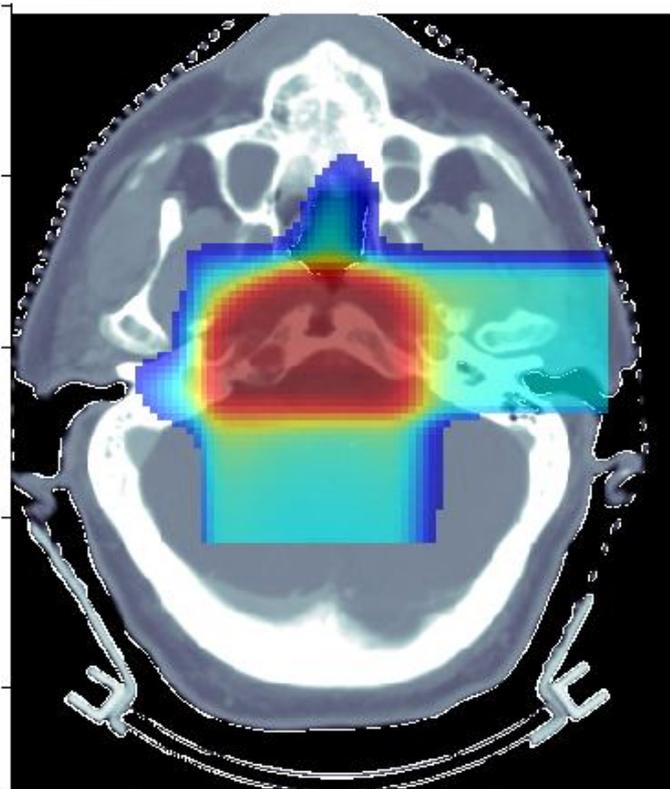


The Gran Sasso in FLUKA

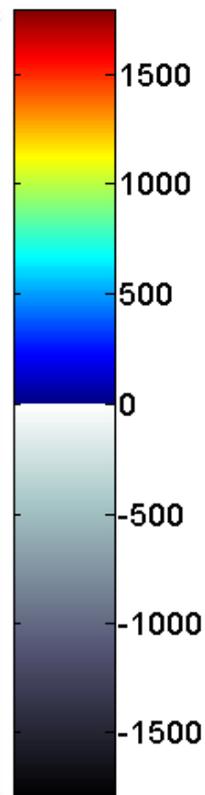
Another example, for medical applications

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

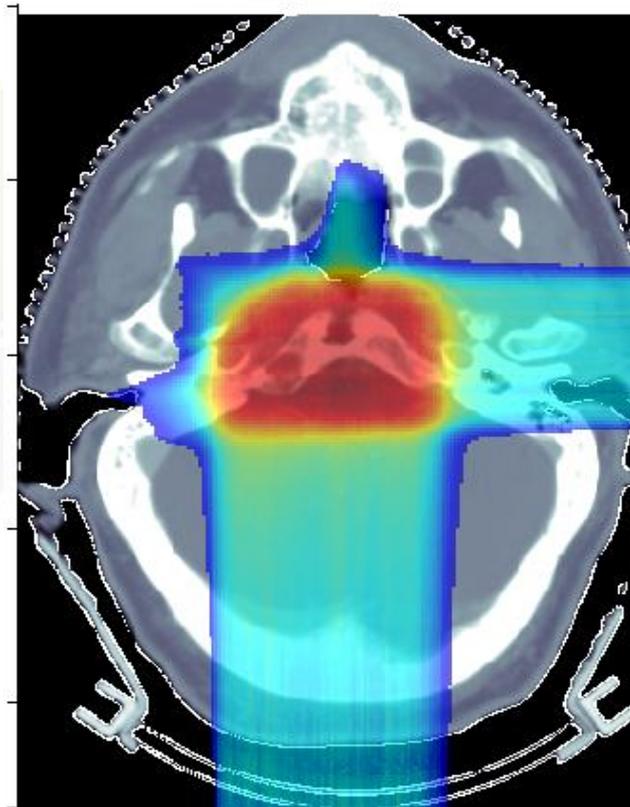
Commercial TPS



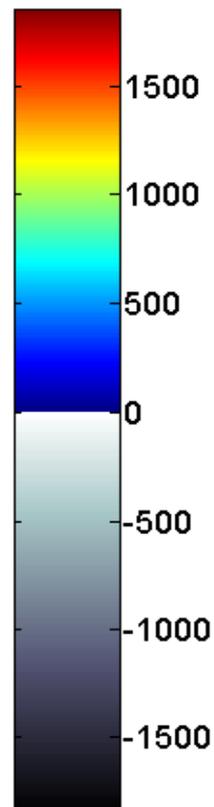
mGy



FLUKA



mGy



Concepts

- A CT scan contains integer values (Hounsfield Unit) reflecting the X-ray attenuation coefficient m_x

$$HU_x = 1000 (m_x - m_{H20}) / m_{H20}$$

- We will use loosely the word “**organ**” to indicate a **group of voxels** (or even more than one group) **made of the same “tissue” material** (same HU value or in a given HU interval)
- The code handles each **organ** as a **CG region**, in addition to other conventional “**non-voxel**” regions defined by the user (the voxel structure is complemented by parts written in the standard combinatorial geometry)
- The code assumes that the voxel structure is contained in a parallelepiped. This RPP is automatically generated from the voxel information.

Procedure (I)

- To describe a voxel geometry, the user must convert his CT scan or equivalent data to a format understood by FLUKA
- This stage should :
 - Assign an organ index to each voxel. In many practical cases, the user will have a continuum of CT values (HU), and may have to group these values in intervals
 - Each organ is identified by a unique integer ≤ 32767 . The organ numbering does not need to be contiguous (i.e. “holes” in the numbering sequence are allowed).
 - One of the organs must have number **0** and plays the role of the medium surrounding the voxels (usually vacuum or air).
 - The user assigns to each NONZERO organ a voxel-region number. The voxel-region numbering has to be contiguous and starts from 1.

Procedure (II)

- The information is input to FLUKA through a special **unformatted** file *.vxl containing:
 - The number of voxels along each coordinate axis
 - The number of voxel-regions, and the maximum organ number
 - The voxel dimension along each coordinate axis
 - A 3D matrix specifying the organ to which each voxel corresponds in Fortran list-oriented format, with the x coordinate running faster than y, and y running faster than z.

val(1)	corresponds to 1,1,1 == organ # of first voxel
...	...
val(N _x)	corresponds to N _x ,1,1
val(N _x +1)	corresponds to 1,2,1
...	...
val(2*N _x)	corresponds to N _x ,2,1
...	...
val(N _y *N _x)	corresponds to N _x ,N _y ,1
...	...
val(N _z *N _y *N _x)	corresponds to N _x ,N _y ,N _z == organ # of last voxel

- A list giving the voxel-region number corresponding to each organ

wrect.f

```
PROGRAM WRTECT
  IMPLICIT DOUBLE PRECISION ( A-H, O-Z )
  * COLUMNS: FROM LEFT TO RIGHT
  * ROWS: FROM BACK TO FRONT
  * SLICES: FROM TOP TO BOTTOM
  PARAMETER ( DX = 2.0D+00 )
  PARAMETER ( DY = 3.0D+00 )
  PARAMETER ( DZ = 4.0D+00 )
  PARAMETER ( NX = 20 )
  PARAMETER ( NY = 20 )
  PARAMETER ( NZ = 20 )
  DIMENSION CT(NX,NY,NZ)
  INTEGER*2 CT
  DIMENSION VXL(NX,NY,NZ)
  INTEGER*2 VXL
  CHARACTER TITLE*80
  DIMENSION IREG(1000), KREG(1000)
  INTEGER*2 IREG, KREG

  *
  CALL CMSPPR
  DO IC = 1, 1000
    KREG(IC) = 0
  END DO
  OPEN(UNIT=30,FILE='ascii_ct',STATUS='OLD')
  READ(30,*) CT
  *
  *
  NO=0
  MO=0
```

Number and
Dimensions
of voxels

read the original CT scan

In this example, the
organ number is simply
set equal to the CT
number for each voxel

For each voxel

```
DO IZ=1,NZ
  DO IY=1,NY
    DO IX=1,NX
      IF (CT(IX,IY,IZ) .GT. 0) THEN
        IO= CT(IX,IY,IZ)
        VXL(IX,IY,IZ) = IO
        MO = MAX (MO,IO)
        DO IR=1,NO
          IF (IREG(IR) .EQ. IO) GO TO 1000
        END DO
        NO=NO+1
        IREG(NO)=IO
        KREG(IO)=NO
        WRITE(*,'(A,2I10)') 'New number, old number: ', NO, IO
      CONTINUE
    END IF
  END DO
END DO
* NO = number of different organs
* MO = max. organ number before compacting
*
WRITE(*,*) ' NO,MO',NO,MO
OPEN(UNIT=31,FILE='ct.vxl',STATUS='UNKNOWN',FORM='UNFORMATTED')
TITLE = 'Egg-like CT scan'
WRITE(31) TITLE
WRITE(31) NX,NY,NZ,NO,MO
WRITE(31) DX,DY,DZ
WRITE(31) VXL
WRITE(31) (KREG(IC),IC=1,MO)
STOP
END
Write the file for FLUKA
or CALL OAUXFI ('ct.vxl',31,'UNKNOWN UNFORMATTED',IERR)
```

Assign organ
IO to this
voxel

If new organ: assign new
region NO to organ IO

Write the file for FLUKA

Input file: geometry description (I)

- Prepare the usual FLUKA input file. The geometry must be written like for a normal Combinatorial Geometry input (in any of the allowed formats, as part of the normal input stream or in a separate *.geo file), but in addition must include:

- **VOXELS** card as the first line, before the Geometry title card, with the following information:

WHAT(1), WHAT(2), WHAT(3) = x, y, z coordinates chosen as the origin of the “**voxel volume**”, i.e. the corner of a RPP extending from **WHAT(1)** to **WHAT(1) + NX*DX**, ... and containing all the voxels

WHAT(4) = possible **ROT-DEFI transformation applying to the RPP**

(**WHAT(5), WHAT(6)** not used)

SDUM = name of the voxel file

(extension will be assumed to be **.vxl**)

```
VOXELS    -20.0    -30.0    -40.0  transf                ct
```

Input file: geometry description (II)

One will have

- The usual list of **NB bodies**, not including the **RPP** corresponding to the “**voxel volume**” (see **VOXELS** card above). This **RPP** will be generated and added automatically by the code as the $(\mathbf{NB}+1)^{\text{th}}$ body, with one corner in the point indicated in the **VOXELS** card, and dimensions **NX*DX**, **NY*DY** and **NZ*DZ** as read from the voxel file.
- The usual list of **NR regions**, with the space occupied by the body named **VOXEL** or numbered **NB+1** (the “**voxel volume**”) subtracted. In other words, the **NR** listed regions must cover the whole available space, except the space corresponding to the “**voxel volume**”. This is easily obtained by subtracting the body **VOXEL** (or **NB+1**) in the relevant region definitions, even though this body is not explicitly input at the end of the body list.

* vacuum inside

```
VACI      5 +SHI +SHTB -SHBT -VOXEL
```

Voxel Regions

The code will automatically generate $NO+2$ additional regions, where NO = number of non-zero organs:

Name	Number	Description
VOXEL	NR+1	sort of a “cage” for all voxels. Nothing should ever be deposited in it. The user shall assign VACUUM to it.
VOXEL001	NR+2	containing all voxels belonging to organ number 0. There must be at least 2 of such voxels, but in general they should be many more. Typical material assignment to this region is air
VOXEL002	NR+3	corresponding to organ 1
VOXEL003	NR+4	corresponding to organ 2
VOXEL###	NR+2+NO	corresponding to organ NO

Voxel Material Assignment

The assignment of materials shall be made by the card **ASSIGNMA**t (and in a similar way for other region-dependent options) referring to the first **NR** regions in the usual way, and to the additional voxel regions using the correspondence to organs.

	ASSIGNMA	BLCKHOLE	BLKH
	ASSIGNMA	VACUUM	VACO
	ASSIGNMA	ALUMINUM	AL
	ASSIGNMA	VACUUM	VACI
cage	ASSIGNMA	VACUUM	VOXEL
Organ 0	ASSIGNMA	VACUUM	VOXEL001
6 "Non-zero" organs	ASSIGNMA	TITANIUM	VOXEL002
	ASSIGNMA	AIR	VOXEL003
	ASSIGNMA	COPPER	VOXEL004
	ASSIGNMA	CALCIUM	VOXEL005
	ASSIGNMA	CARBON	VOXEL006
	ASSIGNMA	AIR	VOXEL007

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