ULYSSES

Particle tracking in complex geometries

Version 3

Luis Peralta

Lisbon, April 2012

Index

Introduction

Ulysses (or Odysseus) is the the legendary Greek king of Ithaca, hero of the Odyssey and the mythological founder of Lisbon. During the Roman Empire the city was called Olisippo or Ulyssippo. The ULYSSES package for tracking and histogramming has been developed in the University of Lisbon since 2007 by Luis Peralta with contributions of Ana Catarina Farinha and Tiago Ribeiro.

The ULYSSES package is a package, designed to make particle tracking in complex volumes and score the results. The histogramming routines are grouped in a library called ULHISTOS and a dedicated manual is available. The geometry package is divided in the **ullib.f** library containing the routines not meant to be modified by the user and the **main**, **ulgeo** and **ulsource** user routines. An include file **ullib.inc** contains all the ULYSSES common variables. In a normal application the user should not have to access the variables in these commons, but if need the variables therein can be customized for the envisage application. In ULYSSES the set of available volumes can be associated to build complex bodies.

Simulation of radiation transport in material systems involve two different kind of tasks: one concerns the physical processes and the other the particle tracking through the geometry. The second task concerning space displacements, interface crossings, etc, is the purpose for which ULYSSES has been created. From the optimization point of view this is a important part of the program since for complex geometries, the particle tracking can take a large fraction of the simulation time.

The ULYSSES package is written in FORTRAN and some knowledge of the language is necessary for an efficient usage of the program. Please refer to the README file in the ULYSSES library folder for details on the program compilation. In ULYSSES the geometry declaration is made through the **ulgeom** routine and the radiation source is defined in routine **ulsource**. The package come with several geometry and radiation source examples, which can be adapted according to the user needs.

The general package flowchart is presented in the figure 1.

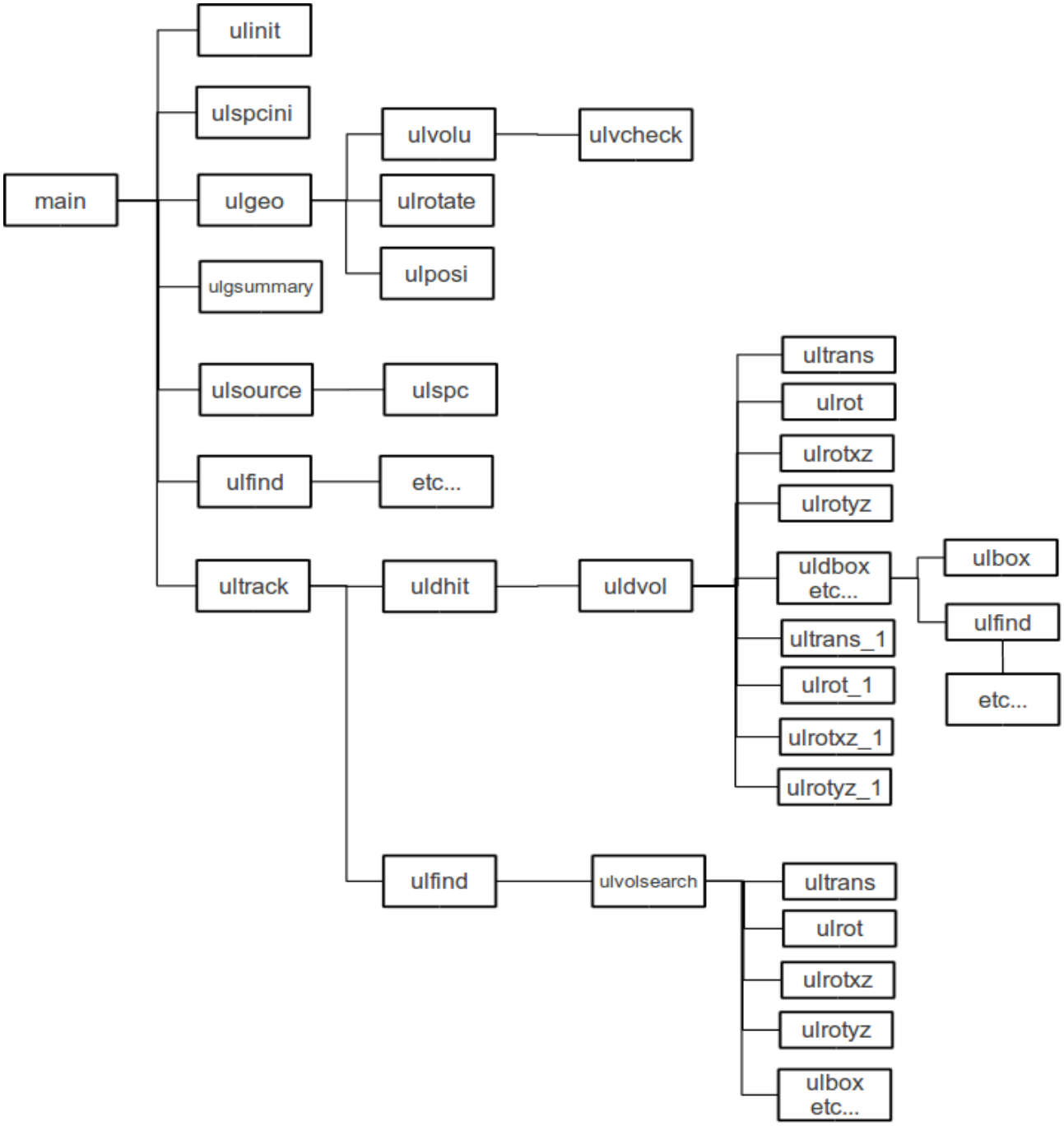


Figure 1. Package flowchart.

The main program

ULYSSES is controlled by a main program (**main.f**) where the call to the interaction with matter routines are made. A call to the routine **ulinit** is mandatory at the beginning of the **main** program and resets all common variables. The optional routine **ulspcini** may be necessary to initialize a spectrum generator.

The histogram book will be made in the beginning of the main program, along with the input of all necessary parameters and variables.

A call to **ulgeo** will then be made to define the setup geometry and a call to **ulgsummary** gives a summary of the input geometry, helping on the geometry debugging. Some tool programs are available for the geometry debugging.

The loop on the primary (and secondary) particles is made following this introductory in this part along with the scoring of the results. Inside this loop the routine **ulsource** is used to generate the primary particle characteristics. The routine **ultrack** finds the next particle position according to a given step. In case of volume boundary crossing the particle is placed very near the boundary but already in the new volume.

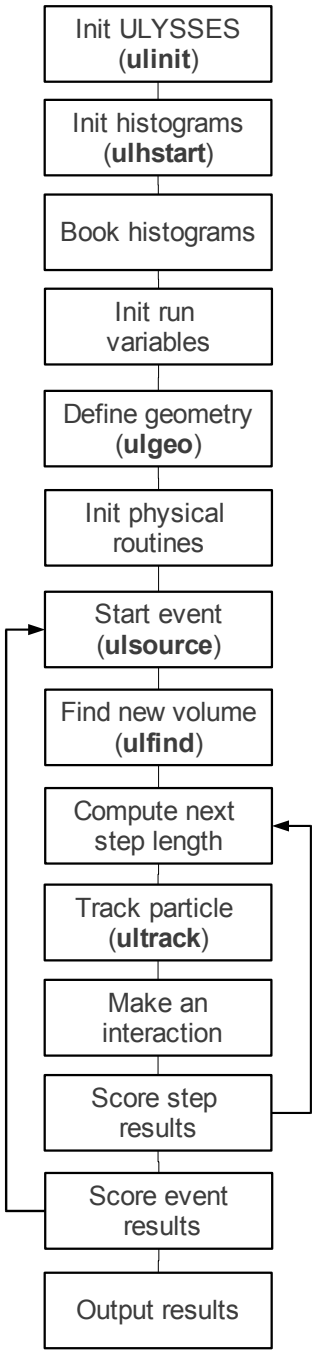


Figure 2: Flowchart of main program.

The common variables of the ULYSSES routines are place in common declarations which are get together in the include file **ulincl.inc** The user usually does not need to access this file. ULYSSES will not need the inclusion of any common variables in the main program,

and the communication with the routines is made by the input/output variables in the called routines. A set of special routines is provided to retrieve some parameters from the program database.

The diagram in the figure 2 resumes the working mechanism inside the **main** program.

Geometry

There are several volumes available in ULYSSES that can be used to build complex bodies. Each volume type has a code number (see table 1). There exists volume sub-types when the volume has some symmetry axis. In these cases there exists type codes for the three main directions (ie x,y,z). These types allow for a CPU time sparring because they use a simpler rotation routine.

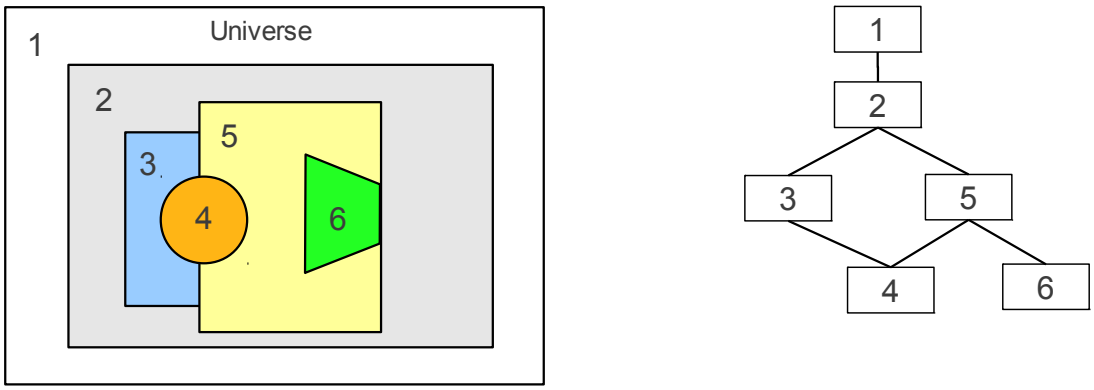


Figure 3: Universe-mother-daughter relationship.

The geometry system used by ULYSSES allows for the construction of rather complex structures by adding (or subtracting) volumes. The volume organization is made using a mother-daughter logic (figure 3). Any volume may have daughter volumes inside. All the volumes have a mother volume except the *universe* volume which contain all volumes. A volume may have more than one mother (shared volume) and that declaration is made in **ulgeom**. The shape of the universe volume is a box and must be big enough to contain all the other volumes. Each volume is identified by its **id** number (1 to 9999). The universe volume has always id=1. Other volumes may have any **id** in the valid range. There isn't any kind of restriction on the order given to the **id** numbers which is free. By definition the universe medium is vacuum (**mat=0**) and a particle falling into the universe volume (i.e. getting out of any other volume) will be discarded (i.e tracking is stopped). As a

consequence, the radiation source can not be directly place inside the universe volume, but instead must be placed within an appropriate material volume.

Each volume has a reference frame attach to it. The main reference frame is the universe reference frame and all operations (rotations and positioning) are referred to it. Other reference frames are defined relative to the universe frame. A volume can be rotated along one of the main (x,y or z) axis, giving the rotation angle. Volumes are created by routine **ulvolume** and then set into position using routine **ulposi** (figure 4). Apart from its **id** a volume **type** must be chosen for each volume (see table 1 and 2) and a material (code) assigned. If the volume is rotated relative to one of the main axis the rotation angle must be supplied. This operation is made via a call to **ulrotate**. The volume's mother id is declared via **ulvolume**. To define a second mother (shared volume) a second call to **ulvolume** should be made. All other parameters in **ulvolume** remain constant (in fact a check is made by the program).

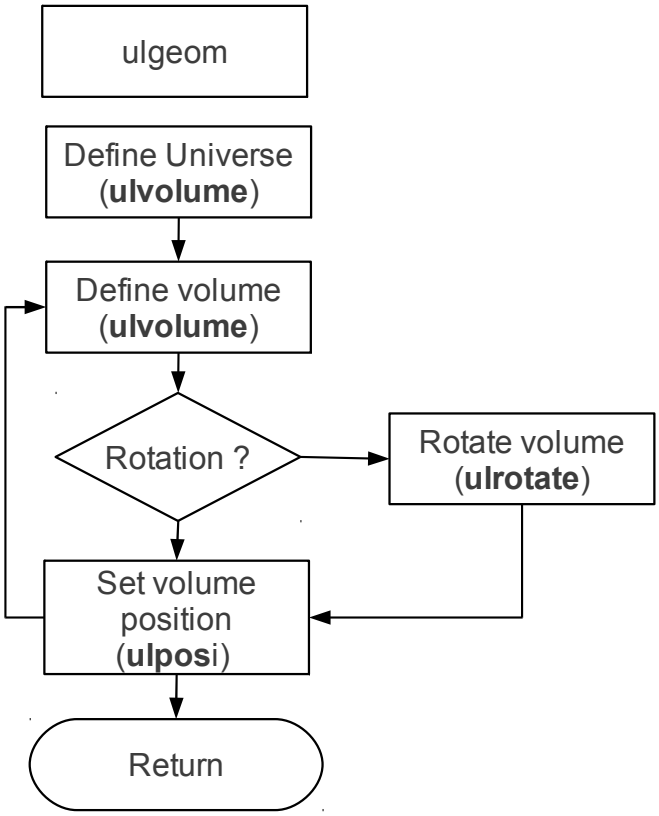


Figure 4: **ulgeom** flowchart.

Radiation source

A source of of radiation has to be defined within the program using the routine **ulsource**. The source emits the primary particle that is going to be followed. The user must specify the particle starting position, direction, kinetic energy and type (a code defined by the user). Many source configurations may be considered, but all the coding is left to the user. In the examples some different sources are given. The source can be extended in space and particles can be emitted in any direction. If the beam is not monochromatic then the user may use routine **ulspc** to generate the energy from a given list of values. This routine can also be used in other cases for instance when the source has different locations (ex. several radioactive sources). If an energy spectrum is available in an histogram produced by ULHISTOS then routine **ulhrnd1** can be used to generate the particle energy.

The routine **ulrndls** can also be used to choose a value among a small list of values. This routine is devised to be used when probability values change, for instance as a function of energy (like the probabilities of the different interaction processes).

Volume codes

Each volume has a 3 number **type** code. The first number is the kind of geometrical figure, the second the sub-type and the third the symmetry axis. In the table below the volume codes are presented.

Table 1: Volume type codes.

Volume	**Type**	**Alignment/ symmetry axis**
Box	100	
Cylinder	200/203 (201) (202)	z (x) (y)
Tube	210/213 (211) (212)	z (x) (y)
Elliptic cylinder	220/223 (221) (222)	z (x) (y)
Sphere	300	
Cut sphere	310/313 (311) (312)	z (x) (y)
Cone	400/403 (401) (402)	z (x) (y)
Pyramid	500/503 (501) (502)	z (x) (y)
Wedge	600/603 (601) (602)	z (x) (y)
Paraboloid	700/703 (701) (702)	z (x) (y)
Ellipsoid	800	z

Volume definition

Each volume is defined by a set of parameters and conditions. All conditions are relative to the volume reference frame. Taking the z axis as the symmetry axis, those parameters and conditions are displayed in the table below. For further explanations on the volume parameters and constrain conditions see the section "Tracking inside the volumes".

Table 2: Volume definition parameters.

Volume	type	Parameters	Conditions
Box	100	1- x side = Lx 2- y side = Ly 3- z side = Lz	$-L_x/2 \le x \le L_x/2$ $-L_y/2 \le y \le L_y/2$ $-L_z/2 \le z \le L_z/2$
Cylinder	200	1- radius = r 2- height = h	$x^2+y^2 \le r^2$ $-h/2 \le z \le h/2$
Tube	210	1-min radius = r min 2-max radius = r max 3- height = h	$r_{min}^2 \le x^2+y^2 \le r_{max}^2$ $-h/2 \le z \le h/2$
Elliptic cylinder	220	1- x axis radius = a 2- y axis radius = b 3- height = h	$x^2/a^2+y^2/b^2 \le 1$ $-h/2 \le z \le h/2$
Sphere	300	1- radius = r	$x^2+y^2+z^2 \le r^2$
Cut sphere	310	1- radius = r 2- min height = hmin 3- max height = hmax	$x^2+y^2+z^2 \le r^2$ $h_{min} \le z \le h_{max}$
Cone	400	1- radius at base = rmax 2- height of cut part= h2 3- cone total height = h	$tg(\theta) = r_{max}/h$ $h-h2 \le z \le h$ $x^2+y^2 \le z^2\, tg^2(\theta)$
Pyramid	500	1- x side = Lx 2- y side = Ly 3- partial height = hmin 4- total height = h	$0 \le z \le h_{min}$ $x_{max} = (h-z)L_x/(2h)$ $y_{max} = (h-z)L_y/(2h)$ $-x_{max} \le x \le x_{max}$ $-y_{max} \le y \le y_{max}$
Wedge	600	1- x side = Lx 2- y side = Ly 3- min height = hmin 4- max height = hmax	$c_x = L_x h_{max}/(h_{max}-h_{min})$ $z_{max} = (c_x - x)h_{max}/c_x$ $0 \le x \le L_x$ $-L_y/2 \le y \le L_y/2$ $0 \le z \le z_{max}$
Paraboloid	700	1- x axis radius = a 2- y axis radius = b 3- height = h	$0 \le z \le h$ $x^2/a^2+y^2/b^2-z \le 0$
Ellipsoid	800	1- x axis radius = a 2- y axis radius = b 3- z axis radius = c	$\frac{x^2}{a^2}+\frac{y^2}{b^2}+\frac{z^2}{c^2}-1 \le 0$

Volume rotations

The volume definition conditions uses the volume local reference frame. If the figure has some sort of symmetry it is assumed the z-axis as the symmetry axis. Figures aligned along the universe x or y-axis must be rotated.

Let S and S' be two reference frames rotated by θ around an axis.

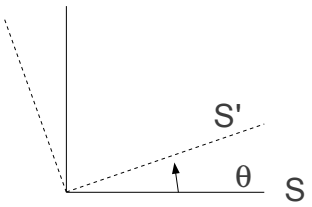


Figure 5: Reference frame rotation.

In the plane a point P with coordinates (u,v) in S will have coordinates (u',v') in S' given by

$$u' = u\cos\theta + v\sin\theta$$
$$v' = -u\sin\theta + v\cos\theta$$

or in matrix format

$$\begin{pmatrix} u' \\ v' \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix}$$

ULYSSES takes the z-axis as the the main symmetry axis. If the volume is aligned along the x or y-axis a rotation of $\theta = -\pi/2$ will be performed on the x-z or y-z plane to change from the universe frame to the volume frame where the volume is aligned along the z-axis.

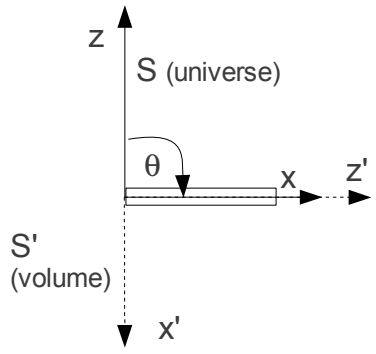


Figure 6 : Rotation to volume reference frame where z'-axis is the symmetry axis.

For this rotation we have

$$\begin{pmatrix} x' \\ z' \end{pmatrix} = \begin{pmatrix} 0 & -1 \\ +1 & 0 \end{pmatrix} \begin{pmatrix} x \\ z \end{pmatrix} \quad .$$

and the inverse matrix is equal to the transpose

$$\begin{pmatrix} x \\ z \end{pmatrix} = \begin{pmatrix} 0 & +1 \\ -1 & 0 \end{pmatrix} \begin{pmatrix} x' \\ z' \end{pmatrix}$$

For a volume align with the y-axis a similar transformation is made, where we simply change x and x' by y and y', respectively. These rotations are performed by the program depending on the volume **irot** value given on calling **ulrotate**.

Rotation about an axis

The volumes can be rotated around one of the main x,y or z-axis. A flag **irot** can be set with the following values on calling routine **ulrotate** in **ulgeom.**

irot=0 : no rotation

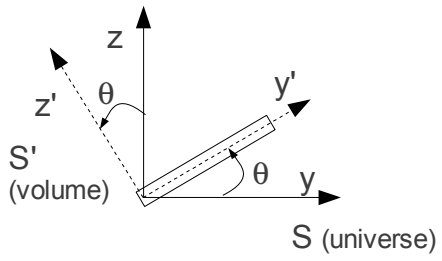
irot=1 : rotation around xx

irot=2 : rotation around yy

irot=3 : rotation around zz

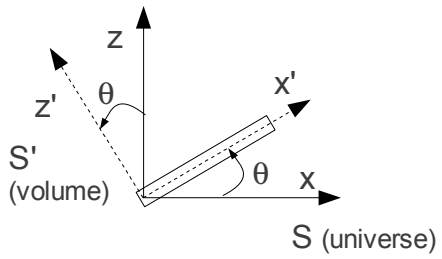
The rotation angle θ is measured relative to one of the universe reference frame axis so that it can always be taken as a rotation on a plane. This is a constrain on the possible rotations, but it is fitted for most practical cases.

irot =1: Volume rotated around the **x-axis** by θ with respect to the **y-axis**



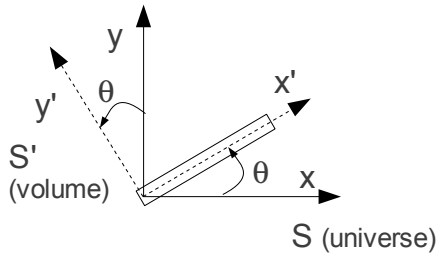
$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

irot =2: Volume rotated around the **y-axis** by θ with respect to the **x-axis**



$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

irot =3: Volume rotated around the **z-axis** by θ with respect to the **x-axis**



$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Tracking inside the volumes

The tracking routine **ultrack** transports the particle through the geometry watching out for boundary crossing. This routine is called in the **main** program. The position $\vec{x}_0$, direction $\vec{u}$, step and current volume **id** are given as input. At exit, the routine outputs the new volume **id** and material index (**mat**).

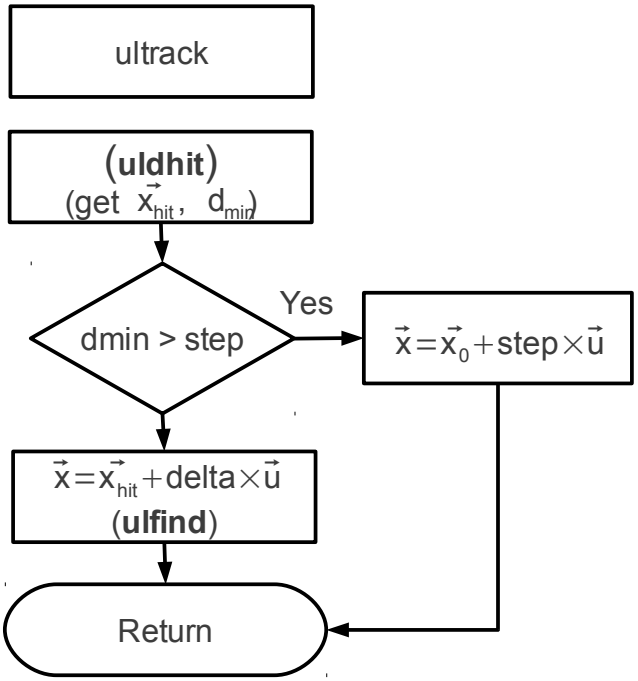


Figure 7: ULTRACK flowchart

Routine **ultrack** calls on **uldhit** routine to compute the nearest hit point with a boundary and the corresponding distance **dmin**. If the distance **dmin** is greater than **step** then a boundary will not be crossed in this call and the particle is transported to a new position $\vec{x}$ according to the equation $\vec{x}=\vec{x_0}+\text{step}\times\vec{u}$.

On the other hand, if a boundary is going to be crossed, the particle is placed across the boundary at the point $\vec{x}=\vec{x_{hit}}+\text{delta}\times\vec{u}$ where $\vec{x_{hit}}$ is the boundary hit point and **delta** a very small quantity (1.0×10^{-10} cm).

Routine **uldhit** uses the routine **uldvol** to compute the hit points and distances to the volume boundaries as well as to the child volumes inside that volume.

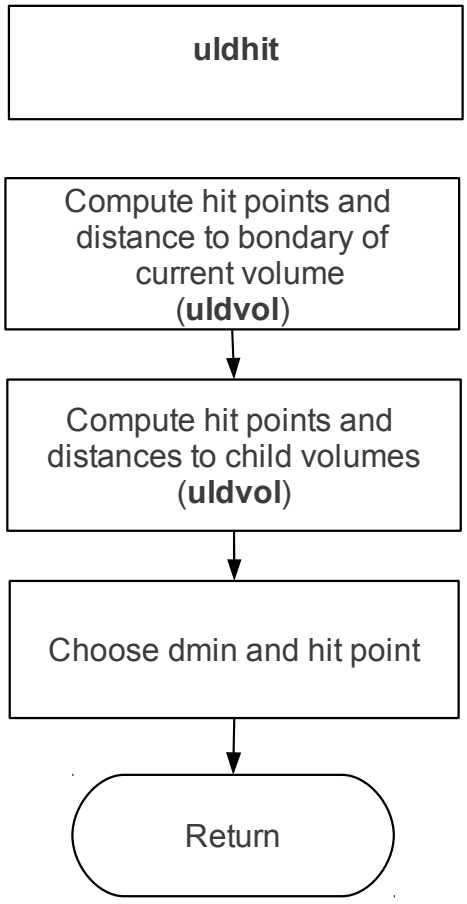


Figure 8: **uldhit** flowchart.

Volume definition and hit points

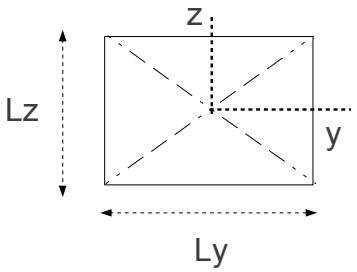
Particle line of flight

Between two interactions the particle line of flight is assumed to be a straight line. Given a starting point $\vec{x}_0$ and a direction vector $\vec{u}$ the vectorial equation of the particle path is define as $\vec{x}=\vec{x}_0+\lambda\vec{u}$ or in components

$$x=x_0+\lambda u_x$$
$$y=y_0+\lambda u_y$$
$$z=z_0+\lambda u_z$$

where λ is a real parameter.

Box



A box with sides of length Lx, Ly, Lz is define by a set of 6 planes given by the equations

$$x=\frac{-L_x}{2},\ x=\frac{L_x}{2},\ y=\frac{-L_y}{2},\ y=\frac{L_y}{2},\ z=\frac{-L_z}{2},\ z=\frac{L_z}{2}\ .$$

A point is inside the box if

$$-L_x/2\le x\le L_x/2$$
$$-L_y/2\le y\le L_y/2$$
$$-L_z/2\le z\le L_z/2$$

Hit points

Planes $x=\frac{-L_x}{2}, x=\frac{L_x}{2}$, if $u_x\neq 0$

$\lambda_1=\frac{-L_x/2-x_0}{u_x}$ and $\lambda_2=\frac{+L_x/2-x_0}{u_x}$

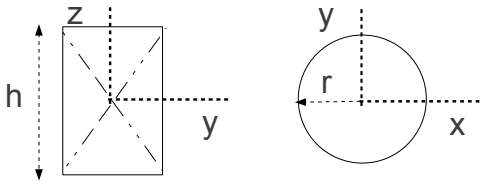
Planes $y=\frac{-L_y}{2}, y=\frac{L_y}{2}$, if $u_y \neq 0$

$\lambda_3=\frac{-L_y/2-y_0}{u_y}$ and $\lambda_4=\frac{+L_y/2-y_0}{u_y}$

Planes $z=\frac{-L_z}{2}, z=\frac{L_z}{2}$, if $u_z \neq 0$

$\lambda_5=\frac{-L_z/2-z_0}{u_z}$ and $\lambda_6=\frac{+L_z/2-z_0}{u_z}$

Cylinder



We define the cylinder side by the equation $x^2+y^2-r^2=0$ and the bottom and top by $z=-h/2$ and $z=+h/2$

A point is inside the cylinder if

$-h/2 \leq z \leq h/2$
$x^2+y^2-r^2 \leq 0$

Hit points

The particle path is defined by a local straight line with vectorial equation

$\vec{x}=\vec{x_0}+\lambda\vec{u}$ or

$x=x_0+\lambda u_x$
$y=y_0+\lambda u_y$
$z=z_0+\lambda u_z$

Top side

$z=+h/2$ and $\lambda=(h/2-z_0)/u_z$

Bottom side

$z=-h/2$ and $\lambda=(-h/2-z_0)/u_z$

Sides

$(x_0+\lambda u_x)^2+(y_0+\lambda u_y)^2-r^2=0$ or

$$\lambda^2\left(u_x^2+u_y^2\right)+\lambda\left(2x_0u_x+2y_0u_y\right)+x_0^2+y_0^2-r^2=0$$

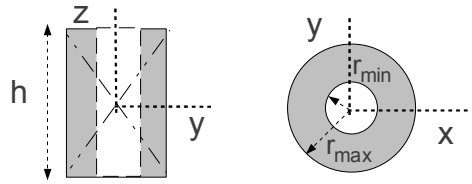
Making

$A=\left(u_x^2+u_y^2\right)$
$B=\left(2x_0u_x+2y_0u_y\right)$
$C=x_0^2+y_0^2-r^2$

we get the values λ solving the 2nd degree equation

$$\lambda=\frac{-B\pm\sqrt{B^2-4AC}}{2A}$$

Tube

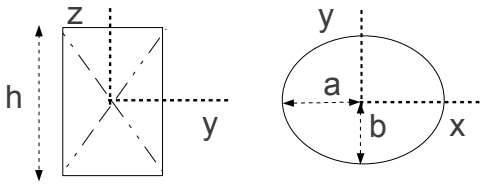


A tube is made of two concentric cylinders. For a tube with length h, internal radius r_{min} and external radius r_{max} an internal point obeys the conditions

$$-h/2 \le z \le h/2$$
$$x^2+y^2-r_{min}^2 \ge 0$$
$$x^2+y^2-r_{max}^2 \le 0$$

The equations for the determination of the hit points are the same as for the cylinder.

Elliptic Cylinder



We define the elliptic cylinder side by the equation

$$\frac{x^2}{a^2}+\frac{y^2}{b^2}=1$$

and the bottom and top by $z=-h/2$ and $z=+h/2$

A point is inside the cylinder if

$$-h/2 \le z \le h/2$$
$$\frac{x^2}{a^2}+\frac{y^2}{b^2}-1 \le 0$$

Hit points

Top and bottom the same as the cylinder.

Sides

$$\frac{(x_0+\lambda u_x)^2}{a^2}+\frac{(y_0+\lambda u_y)^2}{b^2}-1=0$$

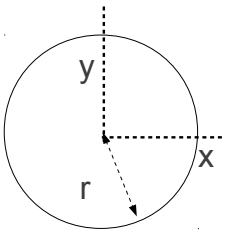
$$\lambda^2\left(\frac{u_x^2}{a^2}+\frac{u_y^2}{b^2}\right)+\lambda\left(\frac{2x_0u_x}{a^2}+\frac{2y_0u_y}{b^2}\right)+\frac{x_0^2}{a^2}+\frac{y_0^2}{b^2}-1=0$$

and defining

$$A=\left(\frac{u_x^2}{a^2}+\frac{u_y^2}{b^2}\right)$$
$$B=\left(\frac{2x_0u_x}{a^2}+\frac{2y_0u_y}{b^2}\right)$$
$$C=\frac{x_0^2}{a^2}+\frac{y_0^2}{b^2}-1$$

we get the values λ solving the 2nd degree equation.

Sphere



The sphere is defined by the equation

$$x^2+y^2+z^2-r^2=0$$

and a point is inside the sphere if $x^2+y^2+z^2-r^2\leq 0$

Hit points

The intersection between the line of flight and the sphere is given by

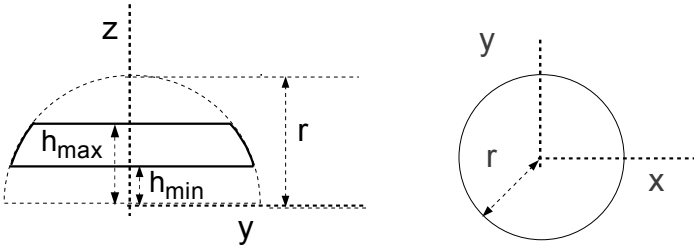
$$(x_0+\lambda u_x)^2+(y_0+\lambda u_y)^2+(z_0+\lambda u_z)^2-r^2=0$$

which a 2nd degree equation with the parameters

$$A=\left(u_x^2+u_y^2+u_z^2\right)=1$$
$$B=\left(2x_0u_x+2y_0u_y+2z_0u_z\right)$$
$$C=x_0^2+y_0^2+z_0^2-r^2$$

and the values of λ are obtained by solving the 2nd degree equation.

Cut-Sphere



The cut-sphere side is defined by the equation $x^2+y^2+z^2-r^2=0$

and the bottom and top parts by $z=h_{min}$ and $z=h_{max}$.

A point is inside the cut-sphere if $x^2+y^2+z^2-r^2\leq 0$, $z\geq h_{min}$ and $z\leq h_{max}$.

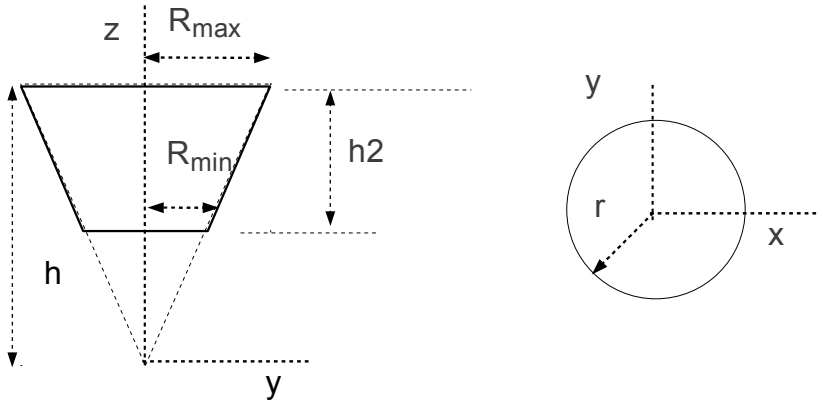
The intersection between the line of flight and the cut-sphere side is given by

$$(x_0+\lambda u_x)^2+(y_0+\lambda u_y)^2+(z_0+\lambda u_z)^2-r^2=0$$

and the solution is obtain in the same way as the sphere.

For the bottom and top sides, i.e. planes $z=h_{min}$ and $z=h_{max}$, if $u_z\neq 0$ the λ is given by $\lambda=\frac{h_{min}-z_0}{u_z}$ or $\lambda=\frac{h_{max}-z_0}{u_z}$.

Cone



The revolution cone surface is define by the equation (vertex at (0,0,0))

$\frac{r^2}{z^2}=tg^2(\theta)$ where θ is the cone opening angle. This equation can be written as

$x^2+y^2=z^2tg^2(\theta)$.

Using the r_{max} and h values we can compute $tg(\theta)$ as $tg^2(\theta)=\frac{r_{max}^2}{h^2}$.

The hit points between particle line $\vec{x}=\vec{x_0}+\lambda\vec{u}$ and the revolution cone are given by

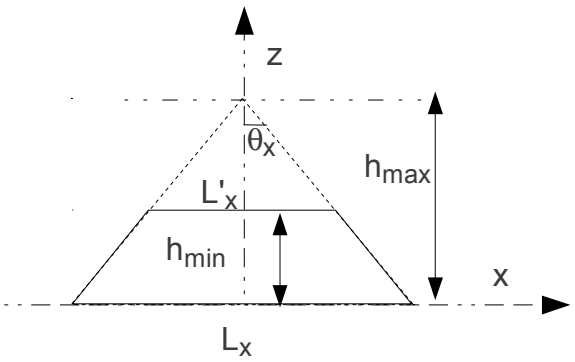
$$(x_0+\lambda u_x)^2+(y_0+\lambda u_y)^2=(z_0+\lambda u_z)^2tg^2(\theta)$$

So one gets a 2nd degree equation $a\lambda^2+b\lambda+c=0$ with

$$a=u_x^2+u_y^2-u_z^2tg^2(\theta)$$
$$b=2(x_0u_x+y_0u_y-z_0u_ztg^2(\theta))$$
$$c=x_0^2+y_0^2-z_0^2tg^2(\theta)$$

and the values of λ are obtained by solving the 2nd degree equation.

Pyramid



Let's consider a pyramid with sides at base L_x and L_y and height h_{max}, and the origin of the coordinate system (0,0,0) at the geometrical center of the rectangular base. The angle between the pyramid and the z axis is θ_x in the x direction and θ_y in the y direction.
The following relations hold between the variables

$$\frac{L_x}{h_{max}} = \frac{L'_x}{h_{min}} \quad , \quad tg(\theta_x) = \frac{L_x}{2h_{max}}$$

$$\cos(\theta_x) = \frac{h_{max}}{\sqrt{h_{max}^2 + (L_x/2)^2}} \quad , \quad \sin(\theta_x) = \frac{L_x}{2\sqrt{h_{max}^2 + (L_x/2)^2}}$$

and similar relation can be written for y.

For the cut pyramid a point (x,y,z) is inside if

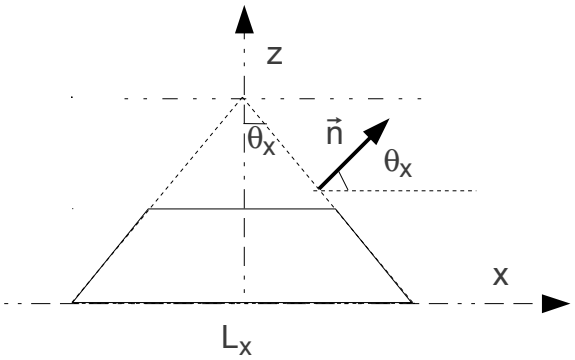
$0 \le z \le h_{min}$ and

$$-L'_x/2 \le x \le L'_x/2$$
$$-L'_y/2 \le x \le L'_y/2$$

where

$$L'_x = L_x \frac{h_{min}}{h_{max}} \quad , \quad L'_y = L_y \frac{h_{min}}{h_{max}}$$

Side plane equation



Let $\vec{n}$ be normal vector to the slanted right side. Then its components are

$$n_x = \cos(\theta_x)$$
$$n_y = 0$$
$$n_z = \sin(\theta_x)$$

and the components of the normal vector at the left side

$$\begin{aligned} n_x &= -\cos(\theta_x) \\ n_y &= 0 \\ n_z &= \sin(\theta_x) \end{aligned} \quad .$$

Each of the four pyramid sides belong to a plane defined by the equation

$$\vec{n}\cdot(\vec{r}-\vec{r}_0)=0$$

where $\vec{r}=(r_x,r_y,r_z)$ is the position vector of any point on that plane.

The common point to the four sides is the pyramid vertex $\vec{r}_0=(r_{x0},r_{y0},r_{z0})=(0,0,h_{max})$.

The plane equation is thus

$$n_x(r_x-r_{x0})+n_y(r_y-r_{y0})+n_z(r_z-r_{z0})=0$$

Hit points

The particle path is defined by a local straight line with vectorial equation $\vec{r}=\vec{x}=\vec{x}_0+\lambda\vec{u}$.

Plugging in the plane equation and solving for λ

$$n_x(x_0+\lambda u_x-r_{x0})+n_y(y_0+\lambda u_y-r_{y0})+n_z(z_0+\lambda u_z-r_{z0})=0$$

$$\lambda=-\frac{n_x(x_0-r_{x0})+n_y(y_0-r_{y0})+n_z(z_0-r_{z0})}{n_x u_x+n_y u_y+n_z u_z} \quad .$$

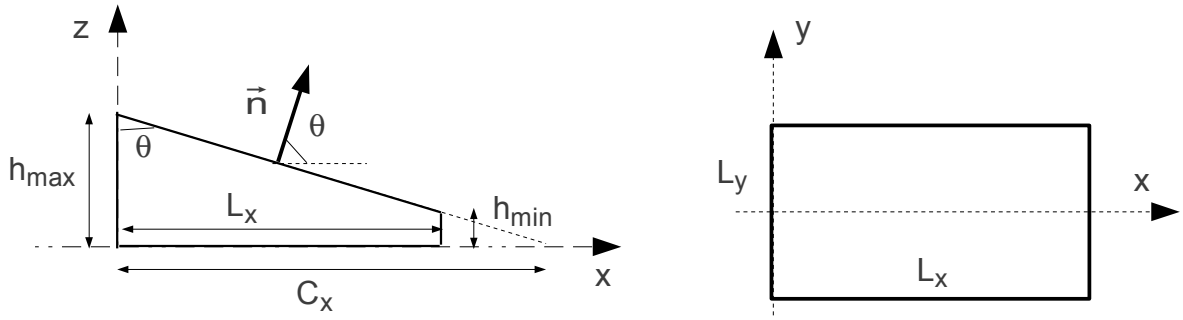
Finally substituting $(r_{x0},r_{y0},r_{z0})=(0,0,h_{max})$

$$\lambda = -\frac{n_x x_0 + n_y y_0 + n_z (z_0 - h_{max})}{n_x u_x + n_y u_y + n_z u_z} \quad .$$

We note that $\vec{n} \cdot \vec{u} = n_x u_x + n_y u_y + n_z u_z$ must be non-zero, i.e. the particle must not travel parallel to the plane, in which case there is no intersection.

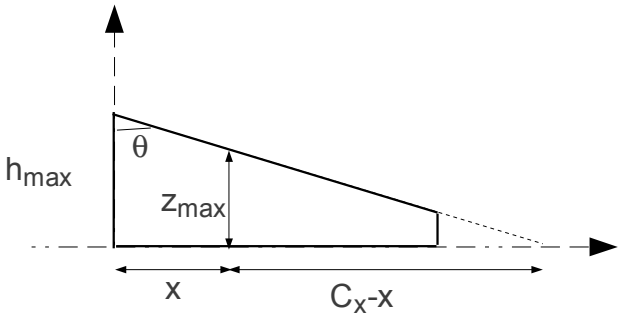
For the bottom and top sides, i.e. planes $z = 0$ and $z = h_{min}$, if $u_z \neq 0$ the λ is given by $\lambda = \frac{-z_0}{u_z}$ or $\lambda = \frac{h_{min} - z_0}{u_z}$.

Wedge



side view bottom view

Let's consider a wedge with sides at base L_x and L_y and heights h_{max} and h_{min} and the origin of the coordinate system (0,0,0) as indicated in the figure. The angle between the slanted side and the z axis is θ.



The length C_x is given by

$$\frac{C_x}{h_{max}}=\frac{C_x-L_x}{h_{min}} \Rightarrow C_x=L_x\frac{h_{max}}{h_{max}-h_{min}} \quad \text{and} \quad h_{max}\neq h_{min} \ .$$

For $0\le x\le L_x$ the following relations hold between the variables

$$\frac{z_{max}}{C_x-x}=\frac{h_{max}}{C_x} \Rightarrow z_{max}=(C_x-x)\frac{h_{max}}{C_x} \quad \text{and}$$

$$\cos(\theta)=\frac{h_{max}}{\sqrt{h_{max}^2+C_x^2}} \ , \quad \sin(\theta)=\frac{C_x}{\sqrt{h_{max}^2+C_x^2}} \ .$$

For the wedge a point (x,y,z) is inside if

$$0\le x\le L_x$$
$$-L_y/2\le y\le L_y/2 \quad \text{and}$$
$$0\le z\le z_{max}$$

Slanted side plane equation

Let $\vec{n}$ be normal vector to the slanted side. Then its components are

$$n_x=\cos(\theta)$$
$$n_y=0$$
$$n_z=\sin(\theta)$$

Each of the four pyramid sides belong to a plane defined by the equation

$$\vec{n}\cdot(\vec{r}-\vec{r}_0)=0$$

where $\vec{r}=(r_x,r_y,r_z)$ is the position vector of any point on that plane.

A point on that plane is $\vec{r}_0=(r_{x0},r_{y0},r_{z0})=(0,0,h_{max})$.

The plane equation is thus

$$n_x(r_x - r_{x0}) + n_y(r_y - r_{y0}) + n_z(r_z - r_{z0}) = 0$$

Hit points

The particle path is defined by a local straight line with vectorial equation $\vec{r} = \vec{x} = \vec{x_0} + \lambda \vec{u}$.

Plugging in the plane equation and solving for λ

$$n_x(x_0 + \lambda u_x - r_{x0}) + n_y(y_0 + \lambda u_y - r_{y0}) + n_z(z_0 + \lambda u_z - r_{z0}) = 0$$

$$\lambda = -\frac{n_x(x_0 - r_{x0}) + n_y(y_0 - r_{y0}) + n_z(z_0 - r_{z0})}{n_x u_x + n_y u_y + n_z u_z} \quad .$$

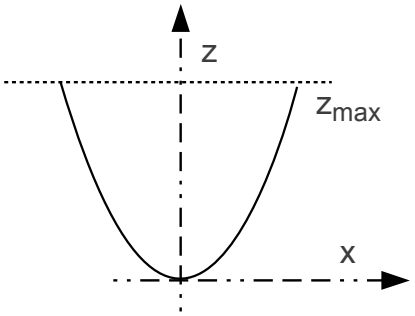
Finally substituting $(r_{x0}, r_{y0}, r_{z0}) = (0,0,h_{max})$

$$\lambda = -\frac{n_x x_0 + n_y y_0 + n_z(z_0 - h_{max})}{n_x u_x + n_y u_y + n_z u_z} \quad .$$

We note that $\vec{n} \cdot \vec{u} = n_x u_x + n_y u_y + n_z u_z$ must be non-zero, i.e. the particle must not travel parallel to the plane, in which case there is no intersection.

For the bottom we have $z = 0$ and for the sides $x = 0$, $x = Lx$, $y = \frac{-Ly}{2}$, $y = \frac{+Ly}{2}$.

Paraboloid



We define the paraboloid by the equation

$$\frac{x^2}{a^2}+\frac{y^2}{b^2}-z=0$$

At z=z_{max} and y=0 we get $x=a\sqrt{z_{max}}$. Likewise at z=z_{max} and x=0 we get $y=b\sqrt{z_{max}}$.
A point is inside the paraboloid if

$$0\le z\le z_{max}$$
$$\frac{x^2}{a^2}+\frac{y^2}{b^2}-z\le 0$$

Hit points

The particle path is defined by a local straight line with vectorial equation $\vec{x}=\vec{x_0}+\lambda\vec{u}$.

Top side z=z_{max} and $\lambda=(z_{max}-z_0)/u_z$

Sides

$$\frac{(x_0+\lambda u_x)^2}{a^2}+\frac{(y_0+\lambda u_y)^2}{b^2}-(z_0+\lambda u_z)=0$$ or

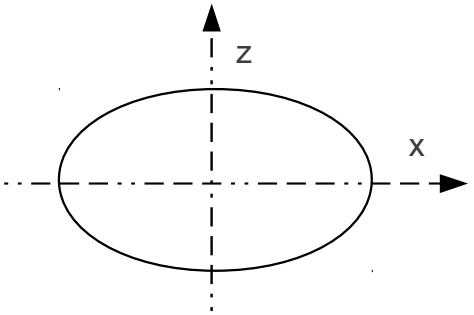
$$\lambda^2\left(\frac{u_x^2}{a^2}+\frac{u_y^2}{b^2}\right)+\lambda\left(\frac{2x_0u_x}{a^2}+\frac{2y_0u_y}{b^2}-u_z\right)+\frac{x_0^2}{a^2}+\frac{y_0^2}{b^2}-z_0=0$$

Making

$$A=\left(\frac{u_x^2}{a^2}+\frac{u_y^2}{b^2}\right)$$
$$B=\left(\frac{2x_0u_x}{a^2}+\frac{2y_0u_y}{b^2}-u_z\right)$$
$$C=\frac{x_0^2}{a^2}+\frac{y_0^2}{b^2}-z_0$$

we get the values λ solving the 2nd degree equation.

Ellipsoid



We define the ellipsoid by the equation

$$\frac{x^2}{a^2}+\frac{y^2}{b^2}+\frac{z^2}{c^2}=1$$

where a,b,c are the axis lengths. A point is inside the ellipsoid if

$$\frac{x^2}{a^2}+\frac{y^2}{b^2}+\frac{z^2}{c^2}-1\le 0 \quad .$$

Hit points

The particle path is defined by a local straight line with vectorial equation $\vec{x}=\vec{x_0}+\lambda\vec{u}$

Sides

$$\frac{(x_0+\lambda u_x)^2}{a^2}+\frac{(y_0+\lambda u_y)^2}{b^2}+\frac{(z_0+\lambda u_z)^2}{c^2}-1=0 \quad \text{or}$$

$$\lambda^2\left(\frac{u_x^2}{a^2}+\frac{u_y^2}{b^2}+\frac{u_z^2}{c^2}\right)+\lambda\left(\frac{2x_0u_x}{a^2}+\frac{2y_0u_y}{b^2}+\frac{2z_0u_z}{c^2}\right)+\frac{x_0^2}{a^2}+\frac{y_0^2}{b^2}+\frac{z_0^2}{c^2}-1=0$$

Making

$$A = \left(\frac{u_x^2}{a^2} + \frac{u_y^2}{b^2} + \frac{u_z^2}{c^2} \right)$$

$$B = \left(\frac{2x_0 u_x}{a^2} + \frac{2y_0 u_y}{b^2} + \frac{2z_0 u_z}{c^2} \right)$$

$$C = \frac{x_0^2}{a^2} + \frac{y_0^2}{b^2} + \frac{z_0^2}{c^2} - 1$$

we get the values λ solving the 2nd degree equation.

User routines

****

ulgeo ()

****

purpose: define the geometry

****

Each volume has an ID number between 1 and 9999. The universe ID is 1 (mandatory). The volumes ID can be any integer number greater than 1, and they don't have any particular order.

****

ulsource (e,x,y,z,u,v,w,kpar)

****

purpose: define a source

output:

e – material id *: real*8*

x,y,z – starting position *: real*8*

u,v,w - direction cosines :*real*8*

kpar – particle type *: integral*4*

****

In this routine the source position and primary particle characteristics are defined. The user must give the starting position (x, y, z) – in cm, direction cosines (u, v, w), particle type (a user defined code) and energy of the primary particle.

Useful routines

****

ulinit()

****

Database initialization. This routine must be the first to be called.

****

ulvolume(id,par,itype,imother,mat)

****

purpose: id confimation

id – identifivation number *: integral*4*

par– volume's parametres *: real*8*

itype – volume's type *: integral*4*

imother – volume's mother id : *integral*4*

mat – material id *: integral*4*

****

The volume organization is made by the number given to each volume in the routine **ulgeo**. This subroutine verifies bad volume identifications and their connection with each other, for example, in the case we have one volume inside another.

****

ulposi(ivolu,xcenter)

****

purpose: id confimation

ivolu *– volume's id : integral*4*

xcenter(3) *- volume's position : real*8*

****

This routine enters the position of volume id=ivolu to the volumes data base.

****

ulrotate(id,irot,angle)

****

purpose: id confimation

id *– volume's id : integral*4*

irot *- rotation type : integral*4*

*angle - rotation angle (deg) : real*8*

****

This routine defines a rotation about one of the x,y or z-axis by an angle.

irot=0 : no rotation

irot=1 : rotation around xx

irot=2 : rotation around yy

irot=3 : rotation around zz

****

ultrack(x,y,z,u,v,w,ibody,stepin,newbody,mat)

****

purpose: track a particle

x,y,z *– starting position : real*8*

u,v,w – direction cosines : *real*8*

ibody *– old volume's id : integral*4*

stepin – step : *real*8*

newbody *– new volume's id : integral*4*

mat *– material id : integral*4*

****

This routine allows the user to track a particle in a complex geometry.

****

ulfind(x,y,z,ibody,mat)

****

purpose: finds volume where particle stands

x,y,z *– starting position : real*8*

ibody *– new volume's id : integral*4*

mat *– material id : integral*4*

****

This routine allows to find the volume to which point (x,y,z) belongs. In the case of volume overlap the routine follows the mother-daughter logic and at the exit ibody is always the daughter volume id.

****

ulhit(x,y,z,u,v,w,id,iflag)

****

purpose: see if volume is hit by particle

x,y,z *– particle position : real*8*

u,v,w *– particle direction : real*8*

id *– volume id : integral*4*

iflag *– hit flag : integral*4*

****

See if particle line of flight intersects volume id. The routine is design to check is a particle keeping the present flight direction will hit a given volume.

****

ulgsummary

****

This routine prints on screen the volume's essential information – number of defined volumes, volume parameters for each volume (volume id, dimensions, center position), number of children, children identification number.

****

ulgetpar(id,par)

****

purpose: uploads the parameter to par

id – volume's id *: integral*4*

par(4)– *volume's parameters array : real*8*

****

This routine gets the volume parameter for a chosen volume.

****

ulput1par(id,ipar,value)

****

purpose: change a volume parameter

id – volume's id *: integral*4*

ipar– *volume parameter : integer*4*

value - *parameter value : real*8*

****

This routines allows the change of a single volume parameter.

On runtime the routine is useful if a variable geometry is to be simulated.

****

ulrndls(P,n,iout)

****

purpose: randomly choses a number from a given list

P- *probability array : real*8*

n- *number of cases in array : integer*4*

iout- *index of chosen value in array : integer*4*

****

The routine randomly chooses a case from a small list according to given probabilities **P**. The output is the position of the chosen case in the array **iout**.

The routine is useful when a case must be chosen from case list with different probabilities, that can modified each time the routine is called.
The probabilities in array **P** don't need to be properly normalized to one, since that step is taken each time the routine is called.

****

ulspcini(id,Ein,Pin,n)

****

purpose: initialize ulspc random number generator

****

id- *generator id number* *: real*8*

Ein- *values array* *: real*8*

Pin- *values probability array* *: real*8*

n- *array dimension* *: integer*4*

This routine must be called prior to **ulspc**. To each generator an **id** number (between 1 and 10) is assigned. The maximum number of generators (10 by default) can be modified in the **ulincl.inc** include file.

****

ulspc(id,Eout)

****

purpose: generate a random number according to given probabilities

****

id- *generator id number* *: real*8*

Ein- *values array* *: real*8*

This routine generates a random number (**Eout**) from a list with given probabilities.
The array of values (**Ein**) and probabilities (**P**) is given in the initialization routine **ulspcini**. The main use of the routine is to generate the primary particle energy, when different energies and intensities are possible. Typical examples are radioactive sources with

several photon energies. The routine can also be used in other cases, for instance when several sources with different positions exist, etc.

Several different generators, with different **ids** are allowed. The routine uses the inverse-transform method to generate the random numbers from the given list.