Monte Carlo simulation of Ion Beam Analysis spectra using Corteo

by François Schiettekatte
Département de physique
Université de Montréal

July 2016

Table of contents

Introduction

Ion beam analysis (IBA) techniques include an ensemble of methods of elemental identification, many of them able to provide quantitative depth profiling of the elements present in the target. (The present document will refer to IBA as the subset of IBA techniques that involve ions both as the incident and detected particle.) From the spectrum obtained, one usually wishes to extract a depth profile. This can be obtained to "first order" by considering an idealization of the technique principles where each incoming ion slows down in straight line in the material until it eventually makes a collision during which it is scattered, produces a recoil or emits a particle in the direction of a point detector. The emitted ion also slows down along a straight trajectory until it leaves the sample. In this idealization, the energy loss is related to the stopping power and the kinematics of the scattering/recoiling collision, so a correspondence between the detected energy and the depth can be established.

To make a step further, IBA practitioners will analyze the spectrum following mainly two ways:

1) Based on the above mentioned approximation, the different parts of an energy spectrum can be iteratively converted into depth profiles, using the composition deduced from the previous iteration to compute the stopping power. This method is only applicable if the different contributions to the spectrum are separable. Furthermore, all contributions to the energy resolution and detection efficiency remain in the resulting depth profile. As a result, an advantage of this method is perhaps that it will not give more information about a sample structure than the data actually contain.
2) One can simulate the energy spectrum (or spectra), usually by representing the sample as a series of slabs of different thicknesses with changing composition, and iteratively modifying (by hand or automatically) the slabs thickness and composition until a satisfying agreement between the data and the simulation is reached. There can be a large number of slabs, the composition changing almost continuously; inferences are then required for the simulation to converge towards a reasonable solution. Other distribution functions than slabs can also be considered. Modern simulation programs usually include in the computation the contribution of effect of different phenomena that affect the spectrum shape. These may include the detector energy resolution, geometrical effects (beam spot and detector size and position), energy straggling, Multiple Scattering (MS) and layer roughness. Other corrections to the spectrum shape can also be included such as compensations for pile-up and pulse height defect. There are mainly two approaches to simulation:
    1. A simulation can be obtained through analytical computation of the spectrum shape. The slabs are converted to trapezoidal shapes obtained by assuming strait line ion trajectories. These shapes are then convoluted for the different contributions mentioned above. It thus requires an analytical expression of the influence of these effects on the spectrum shape. While such expression is available for most effects, some other may be much more problematic to model in some circumstances. This is namely the case of MS with heavy ions or at grazing incidence.
    2. A simulation can be obtained by Monte Carlo (MC) methods. Such simulation encompasses in principle all the important phenomena that affect ion transport by directly simulating ion trajectories. This includes deviations, small or large, form straight-line trajectory, as well as a representation of the detection system closer to the physical one. But the accumulation of a spectrum with sufficient statistics requires the simulation of a large number ($\sim 10^6$) of single ion trajectories, which can be time consuming.

In this document, the later method and its implementation in the form of the program Corteo is described. The document explains in some details what happens during MC simulation of ion trajectories as applied to IBA and what improvements Corteo brings in terms of computation speed efficiency, at the price of what approximations. Its purpose is not to demonstrate if MC is a better solution than analytical simulations. Both have their advantages and inconvenient and both should be part of the IBA practitioner's arsenal. One advantage that Corteo brings over other solutions is that it is entirely open source and distributed under the terms of the Gnu General Public License (GPL),[1] therefore not only usable for free but improvable or adaptable by anyone to fit particular needs. Prior knowledge of IBA is assumed[2], as well as previous (but not necessarily detailed) knowledge of SRIM or TRIM-related programs.[3]

The first chapter of the document introduces MC simulation of ions transport in matter. The second chapter is more specific to Corteo. In order for the user to clearly understand the meaning of the different parameters and the implications of the different approximations, the core of this chapter details the way that calculations are carried out. A description of the input and output files of the simulation program itself is presented in Chapter 3. This is useful for user that would like to control the simulations from outside the user interface, using for instance a scripting language. Finally, instruction about the user interface and tutorials are presented in the forth chapter. Users who are already aware of the meaning of most parameters and want only a quick start may jump directly to chapter 3 and 4, but reading the complete document is suggested for a clear understanding of the parameters meaning and the implications of the different approximations used in Corteo.

The current version of Corteo is intended to simulate Rutherford Backscattering Spectrometry (RBS), Elastic Recoil Detection (ERD) and coincidence. In the case of RBS and ERD, the detector can feature an absorber and can be a Time-of-Flight (TOF) detector. (A TOF detector is actually treated as two subsequent detectors, as in reality.) Nuclear Reaction Analysis is not currently implemented as it would require a knowledge of the cross-section at all kinematic angles. Theoretical values are available for some reaction, so it is planed to implement this kind of analysis in the future.

Corteo also allows the simulation of ion implantation and full collision cascade, but this part of the program won't be discussed here as these types of simulation are still in an early, unverified stage. Use them at your own risk.

1 Transport of ions in matter

The subject of ion transport in matter has been under investigation for more than a century, soon after the discovery of radioactivity. The field benefited from the work of great minds, at a time where particles physics was in the MeV range. At these energies, some approximations can be made, namely about the energy dependence of the stopping power and the screening of the nucleus potential by the electrons. Things get quite more complicated when the ions reach the speed of the electrons. It was however necessary to jump into this kind of difficulty to harness the problem of ion implantation, almost considered as a "black art" before significant improvements to calculation methods where made.

Numerical computation of ion trajectories in materials was already proposed at the end of the 60's by Barrett,[4] and then by Robinson *et al.* a few years later,[5] but involved a lengthy procedure for the computation of ions deflection in a screened potential. Biersack, Haggmark, Littmark and Ziegler made other important steps, namely by introducing a fast method to compute ions deflection (the `MAGIC` algorithm),[6] by devising the ZBL potential that arguably better suites all ion pairs, and by providing a compilation of stopping power data with a model to fit these data and find tendencies to interpolate where experimental data are not available. This work is fully documented in Ref. 3, some of its aspects being summarized below to have a clear understanding of approximations made in our MC simulation.

1.1 Approximations made for the computation of ion transport

In simulations programs related to TRIM, the computation of ion slowdown is carried out under three important approximations: the binary collision approximation (BCA), the central potential approximation (CPA), and the random phase approximation (RPA). The BCA surmises that collisions occur only between two partners, i.e. that the forces between the atoms involved in the collision is much larger than the forces due to the other surrounding atoms. This is an excellent approximation if the minimal approach distance during the collisions is small compared to the inter-atomic distance but it is a very disputable approximation in many circumstances, for example for collisions at low energy or at large impact parameter.

Collisions are assumed to be elastic (total energy conserved) during the slowdown computation, all energy losses related to electronic excitations or ionizations being taken care of separately by the computation of the electronic energy losses. The interaction potential is thus the atomic nucleus potential, but taking into consideration that its charge seems to decrease with increasing distance between collision partners due to their surrounding electrons that screen the nucleus potential. At distances of more than a few Å, the atoms look neutral and unless ionized, interact only through dipole moment (Van der Waals force), which is by all means negligible in our context. Because of the orbitals shape, this screening should include some angular dependence, but this would complexify considerably the computations, so screening functions are most of the time only radially dependent and allow the CPA to be used.

Several screening functions have been proposed. SRIM uses the ZBL potential of the form

$$\Phi(\rho)=\sum_{i=1}^{4} a_i e^{-b_i \rho} \qquad \text{Eq.(1)}$$

where a_i, b_i are parameters chosen to best fit any possible pairs of atoms, and ρ is the atom separation expressed in reduced units (see below). Those unfamiliar with such potential are urged to look into the literature, namely Ref. 3, in order to forge for themselves an opinion about how approximate are these kinds of potentials. We will be bound to this approximation in our computations of MS, often in a regime where the screening function has important effects.

A final approximation made during these calculations is the RPA. Here, it is assumed that the next collision partner is located randomly not only in terms of angle but also in terms of distance from the locus of the last collision, considering a concentration of scattering centers corresponding to the material concentration N. This allows the problem to be treated in the same way as the kinetic theory of gases with molecules described as point objects interacting instantaneously and randomly, with a random flight path between collisions. (Only here, the scattering centers are fixed and the energy decreases proportionally to the flight length of the projectile between collisions due to electronic energy losses.) Considering crystallinely ordered collision partners implies the determination of the next collision partner, a relatively lengthy procedure described in Ref. 5. This is a necessary step, though, if one wants to simulate channeling.

1.2 Computation of ions trajectory

Let the traveling ion have mass M_1, atomic number Z_1 and energy E. For each collision cycle, the steps will be the determination of the flight length, he impact parameter, the collision partner, the scattering and azimuthal angles, and, based on the latter two, of the new flying direction. Then, the energy loss is computed, the position is incremented from the last collision locus to the new one according to the flight length and the new flying direction, and the cycle restarts from the new collision locus until the ion comes to rest or leaves the target.

1.2.1 Flight length

In the context of the RPA, the probability of finding a collision partner in the interval $[\ell, \ell+d\ell]$ follows Poisson's statistics

$$P(\ell)d\ell = e^{-\ell/\ell_0}d\ell \qquad \text{Eq.(2)}$$

so the next flight length is computed as

$$\ell = \ell_0 \ln(-r_1) \qquad \text{Eq.(3)}$$

where r_1 is a random number in the interval]0,1]. In TRIM-related programs, the mean free path ℓ_0 usually corresponds to the inter-atomic distance

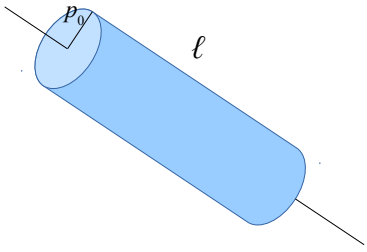
$$\ell_0 = \frac{1}{\sqrt[3]{4\pi N/3}}\,. \qquad \text{Eq.(4)}$$

Therefore, not only the structure of the material is completely disregarded, but collisions may occur at flight lengths smaller than the inter-atomic distance. We will see in Section 3.1.1 that adjusting the flight length is one of the approximations made in Corteo.

1.2.2 Impact parameter

Next, the impact parameter p is selected. Its maximum value p_0 is such that a cylinder of radius p_0 and length ℓ should contain one scattering center, that is,

$$\pi\, p_0^2\, \ell N = 1\,. \qquad \text{Eq.(5)}$$



The probability of finding p being uniformly distributed over the circular section of the cylinder,

$$\pi\, p^2\, \ell N = r_2 \;\Rightarrow\; p = \sqrt{\frac{r_2}{\pi\, \ell\, N}} \qquad \text{Eq.(6)}$$

where r_2 is a random number in the interval [0,1]. Note that if ℓ_0 is chosen to be relatively large, the values of p will be correspondingly smaller, resulting in fewer but larger angle collisions.

1.2.3 Collision partner

In a compound materials, considering the RPA, the next collision partner, of mass M_2 and atomic number Z_2, is found randomly according to the stoichiometry defined by the user, thus involving a third random number r_3.

1.2.4 Scattering and azimuthal angles

One can then compute a reduce impact parameter

$$s = p/a \qquad \text{Eq.(7)}$$

where

$$a = \frac{0.8853\, a_0}{Z_1^{0.23} + Z_2^{0.23}} \qquad \text{Eq.(8)}$$

is the screening length, found in Ref. 6 to best satisfy most atom pairs, and a_0 is Bohr's radius. On can also compute the reduced energy

$$\varepsilon = f\, E_{CM} \qquad \text{Eq.(9)}$$

with

$$f = a/Z_1 Z_2 e^2 \qquad \text{Eq.(10)}$$

and

$$E_{CM} = \frac{E}{\left(\frac{M_1}{M_2} + 1\right)} \qquad \text{Eq.(11)}$$

the energy in the center-of-mass (CM). Thanks to the CPA, using (dimensionless) reduced parameters s and ε, and expressing the distance r in reduced units $\rho = r/a$, one can compute the scattering angle in the CM by solving the following integral:

$$\Theta_{CM}(\varepsilon, s) = \pi - 2\int_{\rho_0}^{\infty} \frac{s \;\; d\rho}{\rho^2 \sqrt{1 - \frac{\Phi(\rho)}{\rho\,\varepsilon} - \left(\frac{s}{\rho}\right)^2}} \qquad \text{Eq.(12)}$$

where ρ_0, the minimal approach distance in reduced units, is solution to

$$1 - \frac{\Phi(\rho_0)}{\rho_0 \varepsilon} = \left(\frac{s}{\rho_0}\right)^2 . \qquad \text{Eq.(13)}$$

Here, the potential is a screened Coulomb potential $\Phi(\rho)/\rho$ with a screening function given for example by Eq. 1. An exact solution to Eq. (12) can be found for an unscreened Coulomb potential, but in our case, the screening cannot be ignored and Eq. (12) has to be solved numerically. Using the Gauss-Mehler quadrature,[14] one has

$$\Theta_{CM}(\varepsilon, s) = \pi - \frac{2\pi s}{M\,\rho_0} \sum_{i=0}^{M/2} H[\cos((2i-1)\pi/2M)] \qquad \text{Eq.(14)}$$

where

$$H[x] = \sqrt{\frac{1 - x^2}{1 - \frac{\Phi(\rho_0/x)}{\varepsilon\,\rho_0/x} - \left(\frac{s}{\rho_0/x}\right)^2}} \qquad \text{Eq.(15)}$$

and M>>1. Such numerical integration can be very long to compute, so Biersack *et al.* developed their `MAGIC` algorithm,[6] an iterative procedure that considerably accelerates the computation of $\sin^2\Theta_{CM}/2$.

This is then used to compute $\sin\Theta_{CM}$ and $\cos\Theta_{CM}$. Still, each iteration in the MAGIC algorithm involves the screening function (Eq. 1), thus requiring the computation of several exponentials. Then, in the laboratory reference frame,

$$\theta = \arctan\left(\sin\Theta_{CM} / \left(\cos\Theta_{CM} + \frac{M_1}{M_2}\right)\right) . \qquad \text{Eq.(16)}$$

Alternatively, in order to minimize the number of complex operations, one has

$$\cos\Theta_{CM} = 1 - 2\sin^2\Theta_{CM}/2 \qquad \text{Eq.(17)}$$

and can obtain

$$\cos\theta = \frac{\cos\Theta_{CM} + M_1/M_2}{\sqrt{1 + 2\rho\cos\Theta_{CM} + (M_1/M_2)^2}} . \qquad \text{Eq.(18)}$$

with only one square root to compute. Now that we have the components of the scattering angle in the laboratory reference frame, we need to determine (randomly in the context of the RPA) the azimuthal angle of the collision

$$\omega = 2\pi r_4 \qquad \text{Eq.(19)}$$

1.2.5 New flight direction

As illustrated in Fig. 2, knowing θ, ω and $\hat{u}=[l,m,n]$ a unit vector parallel to the flight direction prior to the collision, it is shown in Appendix I that the components of the flight direction after the collision, $\hat{u}'=[l',m',n']$, are given by

$$\begin{aligned} l' &= l\cos\theta + \frac{\sin\theta}{\sqrt{1-n^2}}(l\,n\cos\omega + m\sin\omega) \\ m' &= m\cos\theta + \frac{\sin\theta}{\sqrt{1-n^2}}(m\,n\cos\omega - l\sin\omega) \\ n' &= n\cos\theta - \sqrt{1-n^2}\sin\theta\cos\omega \end{aligned} \qquad \text{Eqs. (20)}$$

These are the equation found in the DIRCOS routine of TRIM-related programs.

1.2.6 Energy loss

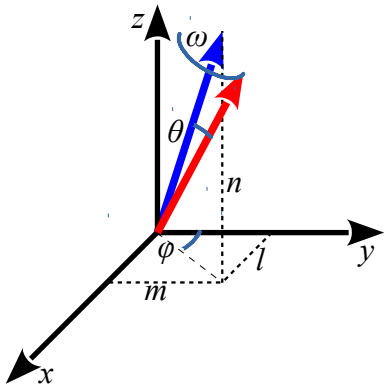
Finally, the ion looses energy elastically during the collision

$$\Delta E_n = \frac{4 M_1 M_2}{(M_1 + M_2)^2} E \sin^2\Theta_{CM}/2 \qquad \text{Eq. (21)}$$

and inelastically along its flight path between collisions

$$\Delta E_e = \left(\frac{dE}{dx}\right)_e \ell + (\Delta E)_{strag}\sqrt{\ell} \qquad \text{Eq. (22)}$$

where the first term on the right hand side represents the electronic energy losses (including an eventual compound correction) and the second term accounts for the energy straggling. Computing the electronic energy losses based on fitting equations found in Ref. 3 also involves several transcendental functions.



1.2.7 Energy straggling

Energy straggling $(\Delta E)_{strag}$ accounts for the fact that electronic energy loss process is stochastic: two ions going through the same

amount of matter do not necessarily excite the same atomic levels or number of electrons and thus they do not loose the same amount of energy. Except at high energy, the probability of exciting an electron depends strongly on ion speed. During each flight ℓ, energy straggling can contribute a positive or negative correction to the energy loss. Assuming that the energy straggling distribution is Gaussian (which is not always the case), one can compute for each flight length

$$(\Delta E)_{strag} = \Omega \, \mathrm{erf}^{-1}(r_5)\sqrt{2} \, . \qquad \text{Eq. (23)}$$

This involves the inverse error function evaluated with a random number $r_5 \in]-1,1[$. Three models are usually considered for Ω:

- Bohr[7], who assumes that all electrons contribute equally to energy straggling, which is a valid assumption at high energy, and amounts to

  $$\Omega^2_{Bohr} = 4\pi Z_1^2 Z_2 e^4 N \, ; \qquad \text{Eq. (24)}$$

- Chu[8] who, based on Hartree-Fock calculations, computed a correction to Bohr's result for protons and α particles at lower energies, giving for the energy straggling

  $$\Omega^2_{Chu} = \Omega^2_{Bohr} / (1 + c_1 E^{c_2} + c_3 E^{c_4}) \qquad \text{Eq. (25)}$$

  where the c_i parameters, listed in Ref. 8, depend on Z_2;

- Yang *et al*.[9] who devised an additional empirical correction to Chu's correction, based on a fit on most available experimental data at the time, resulting in an energy straggling of the form

  $$\Omega^2_{Yang} = \gamma^2 \Omega^2_{Chu} + A_0 \frac{A_1 \Gamma}{(\varepsilon - A_2)^2 + \Gamma^2}, \quad \Gamma = A_3 (1 - e^{-A_4 \varepsilon}) \qquad \text{Eq. (26)}$$

  where, for protons, $\varepsilon = E$ and $A_0 = 1$, while $\varepsilon = E / Z_1^{3/2} Z_2^{1/3}$ and $A_0 = Z_1^{4/3} / Z_2^{1/3}$ for heavier ions, and γ is the ratio of the effective charge of the ion to that of a proton with the same velocity.

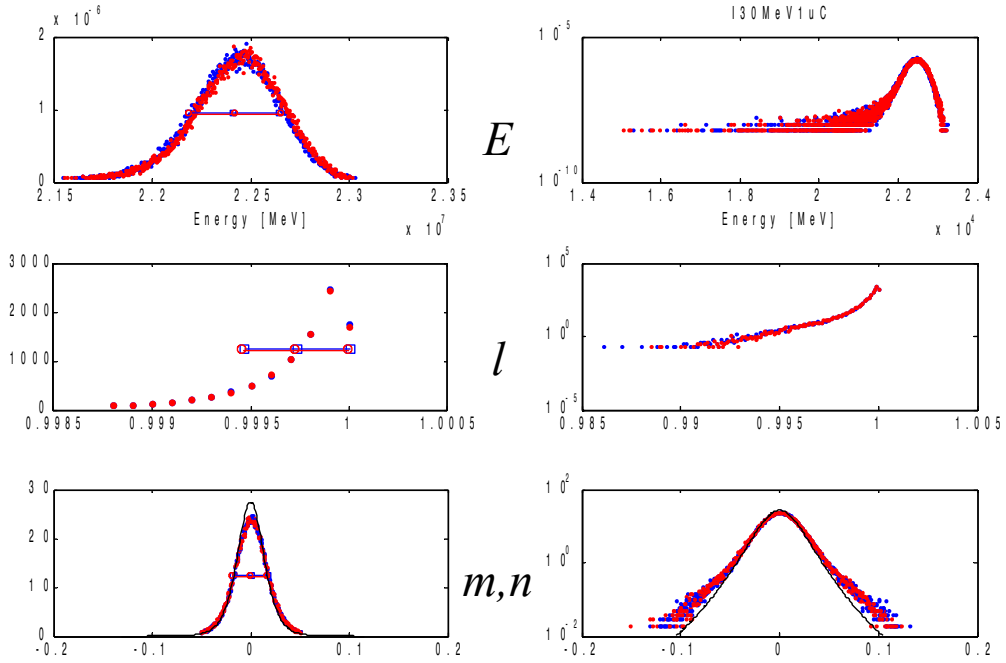
These models are usually also implemented in analytical simulation programs.

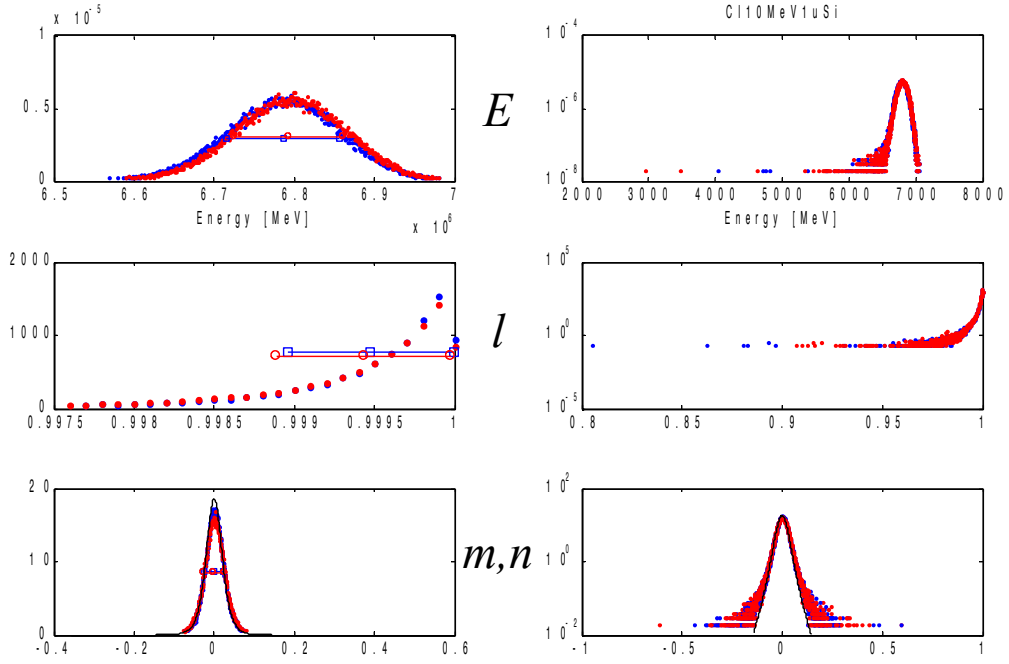
1.3 Multiple scattering

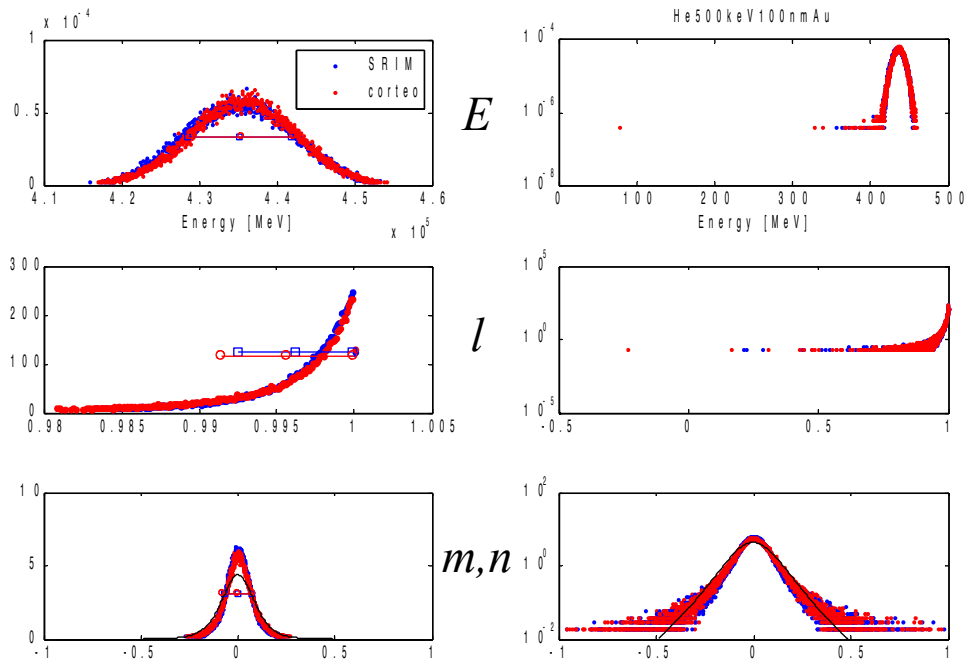
In the context of ion beam analysis, trajectories computation helps to account as precisely as possible for the effect of multiple collisions by reproducing in details the deviations from straight line trajectories. These deviations can be small and translate into a peak broadening, but can also be significant and contribute a broad background to the energy spectrum. The latter is often called plural scattering. But as MC simulates the full process with a continuous transition between the peak broadening effect and the formation of a spectrum background, it intrinsically takes into account MS and there is no non-arbitrary criteria to distinguish between the two regimes. For this reason, in this document, the expression Multiple Scattering designates both multiple and plural scattering without distinction, that is, all collisions at small or large angle that an ion undergoes while traveling into matter. However, as we will see below, Corteo uses an approximation that distinguishes the main scattering event (called the “main collision”) from all the other that occur during the ions slowdown process.

1.3.1 Analytical model for multiple scattering

An analytical model of the angular distribution of ions as a function of thickness has been developed by Sigmund and Winterbon.[10] This model is based on the RPA and CPA, as for MC, and on the additional approximation that the scattering angles are small (they quote “ <20° ”). We could add that the distribution is assumed to be symmetric; this might not be the case if, for example, a surface from where the ions can escape is involved. Assuming Tomas-Fermi or Lenz-Jensen scattering, they


E
l
m,n
Energy [MeV]
Energy [MeV]


E
l
m,n
Energy [MeV]
Energy [MeV]



determined a method to compute the probability of having a certain angular deflection α after crossing a certain reduced thickness τ. This computation involves long summations, so a table is provided from which the probability can be extracted. Szilagyi, Paszti and Amsel (hereafter named the SPA model)[11] developed a model where they fitted projected distributions obtained form the distributions tabulated by Sigmund and Winterbon. The fitting function is a Pearson VII type distribution with its with and exponent depending on τ.

The key aspect to retain from Refs. 10 and 11 is that these broadenings are not Gaussian, or only become Gaussian for large thicknesses. The SPA model is often used by analytical simulation codes to account for small-angle MS.

1.3.2 Comparison to Monte Carlo

Both the SPA model and MC simulations are based on the RPA and CPA, but the MC simulation is not bound by the small angle scattering approximation. A comparison of the angular distributions obtained with the SPA model with those obtained by MC is presented in Figs. 3-5. The bottom row of each figure shows the distribution of the components m and n of the directional vector $\hat{u}$ at the exit of the target for different beam/target combinations obtained from SRIM and Corteo, and compared to the SPA model. (Comparison between Corteo and SRIM is discussed in the next chapter.) m and n are the components perpendicular to the incident beam axis, so they correspond to the sine of the angular deviation. The graphs on the right are plotted on a logarithmic vertical scale. A Gaussian broadening would appear parabolic on this scale. This is clearly not the case, as pointed out by SPA. It is seen that the SPA model reproduces fairly well the spread except for wings visible only on the semi-logarithmic graphs. It is also seen that the energy distributions (top graphs) feature, in addition to the Gaussian broadening due to energy straggling, an asymmetric low-energy tail that is the result of MS. Such tails and wings contribute to the background in energy spectra, but also to peak broadening in a non obvious

way. MC simulations can therefore be useful to give a more accurate account of these effects. Another point to outline about these graphs is that while heavy ions of Fig. 3 have a distribution of their *l* component relatively close to 1 after passing through a light matrix, thus still flying almost in strait line, it is observed in Fig. 5 that light ions suffer a much broader distribution of their directional vector, which extend over almost the full range of their possible values. This means that their trajectories can be much more random after crossing a certain depth, resulting in increasingly disordered trajectories. It is then questionable to assume that the trajectory is strait and to account for the deviations by a broadening of the spectrum peaks. In addition, the fact that such trajectories exist contribute not only to a background signal, but also in a change in the intensity of the spectrum peaks themselves.

Finally, when the distribution becomes broad compared to the beam angle with the sample surface, the distribution is no longer symmetric, as illustrated in Fig. 7 for ions of Fig. 5 but incident at 70°, and cannot be represented accurately by the SPA model. Measurements made at grazing incidence on heavy substrates would therefore benefit from being compared to MC simulations, as illustrated in Fig. 6.

1.4 Other computations

MC simulation programs such as TRIM also have the possibility to follow each collision partner down to a certain energy, called the displacement energy. This can be used to estimate the defect concentration and the physical sputtering yield. While it is also possible to simulate full cascades with Corteo, this part of the code is still under development and unverified (so use it at your own risks), and not necessary even for ERD simulations because of the "main collision" approximation as we will see in the next chapter. Following each recoil until a spectrum is accumulated would make simulations almost infinite.

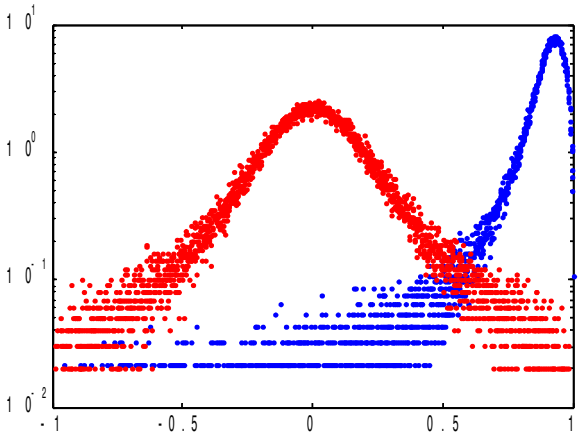


Fig. 7: Distribution of *m, n* components of the directional vector for 500 keV He ions at the exit of a 100 nm Au target when hitting the target at 70° from surface normal.

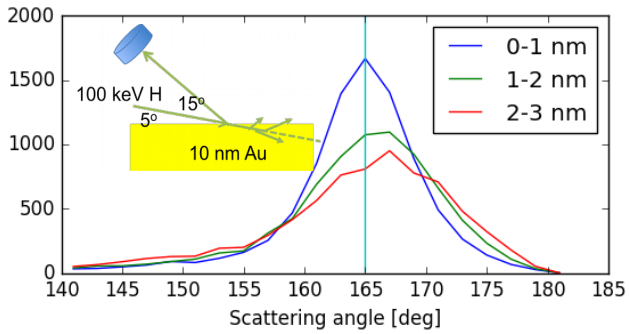


Fig. 6: Distribution of scattering angle for *detected ions* having made their main collision at indicated depths ranges, showing the effect of grazing incidence: 100 keV H incident at 5° from the surface of an Au layer. Detector at 165° from beam axis in IBM geometry. Ions scattered from the first 1 nm are detected after a 165° scattering. However, ions scattered from deeper depth show an increasing scattering angle. This is because ions that suffered MS towards the surface may be lost while those which suffered MS towards depth remain in the target. So on average, the beam "turns away" from the surface, and detected ions have to scatter at larger angles.

2 Computation of an IBA simulation in Corteo

As we have seen in previous chapter, despite important approximations, the number and type of calculations that have to be carried out in order to compute each flight steps along the trajectory of an ion is quite considerable. Simply using TRIM on a PC to simulate an IBA spectrum would reportedly take months or years if no other improvement to computation efficiency than keeping the target thin is included.[12] Other strategies have to be considered, sometimes at the price of additional approximations, in order to bring the simulation time in a more manageable range. In the next section, different approaches to decrease the simulation time and their implementation in Corteo are discussed.

Corteo combines a series of optimizations that, if all applicable, make simulations achievable in a few seconds on a personal computer. The following sections describe how the slowdown of each ion is computed. Following Arstila,[13] scattered ions and/or recoils are deliberately generated from all target depths within a cone towards the detector, and the slowdown of the emitted ion is followed until it reaches the surface or a certain minimum energy. For detection, the concept of "virtual detector" proposed by Arstila is also implemented.

In addition to Arstilla's strategies, Corteo relies on tables stored in memory to extract the flight length, angular components of trajectory rotation, and stopping power and energy straggling values. These "fine-grained" tables are logarithmically distributed thanks to an indexing mechanism based on the digital representation of floating-point numbers. Using other computational strategies, ion slowdown is thus simulated without calling any trigonometric, transcendental or inverse functions, except for one inverse square root, as discussed below. Divisions are also avoided as much as possible.

2.1 Techniques implemented in Corteo

The current version of Corteo implements three IBA techniques: RBS, ERD and coincidence. In the case of RBS and ERD, the detector can feature an absorber and can be a TOF detector. (A TOF detector is treated as two subsequent detectors, as in reality.) Neither NRA nor channeling are currently implemented. NRA would require a knowledge of the cross-section at all kinematic angles. Theoretical values are available for some reaction, so it is planed to implement this kind of analysis in the future. Channeling requires abandoning the RPA and finding the next collision partner deterministically.

Fig. 8 illustrates the implemented techniques. In all cases, the slowdown of the incident ion is simulated down to a certain depth where, during what is called the "main collision" a scattered ion and/or recoil is generated. The slowdown of the emitted ion(s) is computed until it/they reach a surface or minimum energy of the simulation. Slowdowns are computed based on the theory described in the first chapter, but with significant improvements to the computation efficiency as described in the present chapter. In the case of coincidence, both the scattered ion and recoil are

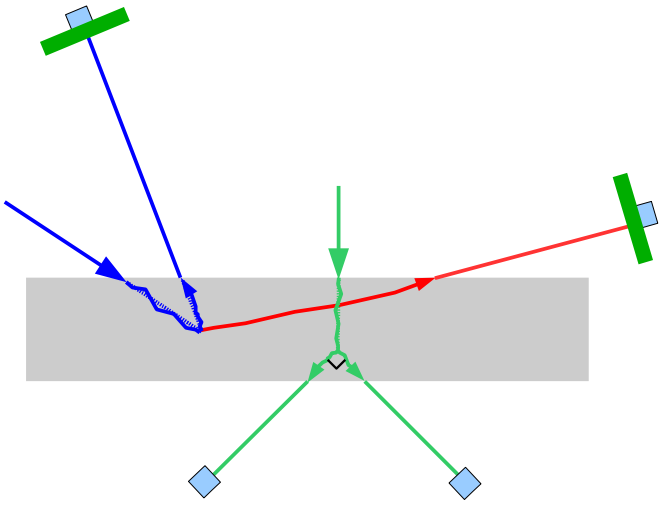


Fig. 8: Illustration of the IBA techniques implemented in Corteo: RBS in blue, ERD in red and coincidence in green. RBS and ERD can feature a virtual and/or a TOF detector. Virtual detectors are not available for coincidence, which must involve identical particles.

followed, but they must be identical (same Z and mass) in order to be able to emit them at 90° one of each other. Virtual detectors are not applicable to this technique because it would involve the correction of both trajectories (since the rotation of one trajectory involves the rotation of the other because the ions are emitted at 90° during the main collision). Moreover, correcting trajectories after virtual detection degrades the quality of the prediction of the effect of MS, and the whole point of simulating coincidence by MC is to precisely simulate the effect of MS on detection efficiency.

2.2 Computing flow of Corteo

Lets detail the computing flow of an IBA simulation with Corteo in order to see where computational improvements are the most necessary and how they are implemented. This will also allows to explain some simulation parameters in their context. From now on, values included in curled braces {} refer to values stored as lists, tables or matrices in memory. A list is a series of values accessed sequentially while the elements of a table (1D) and a matrix (2D) are accessed by first computing the index(es) to get a specific value (e.g. stopping power at a given energy).

0. The target is specified by a set of layers, or a bitmap with a certain number of colors each representing a material. (A substrate layer can also be specified below the bitmap.) Each layer or material (color) have a certain density and contain a number of elements whose fraction of the total is specified. There is in principle no limit* to the number of layer/material and elements in each layer, although larger numbers increase memory usage and computing time. The beam atom, its energy and direction are set, as well as detector coordinates. At this step, Corteo computes several parameters whose purpose will be explained all along, and referred to as precomputed parameters, values, lists, tables and matrices.

   The program then cycles through the partners for the main collision (the detected element) in each layer. (In the case of bitmaps, it is selected randomly according to the composition at the location of the collision, see below.) As mentioned previously, the detected element does not need to be one constituting a layer. If the detected element is only a trace element or isotope, the user can neglect it in the target description, which only serves as a slowing/multiple scattering medium. This will save unnecessary memory usage and accelerate slowdown computation.

The simulation then goes as follow:

1. Ion slowdown: an incident ion starts at surface with energy E, position $\vec{r}$ and directional vector $\hat{u}$ set by the beam parameters. In the case of a layered sample, that position is the origin. In the case of a 2D or 3D bitmap, the point is selected randomly at the top of the bitmap. We now examine the slowdown process. Each ion might be subject to hundreds of collisions, and we may simulate millions of ions, so this section of the computing flow may run billions of times and must be very computationally efficient.
   a. The program checks, according to ion current depth x, what is the current layer, or material at position x,y,z in the case of a bitmap.
   b. The ion changes direction as a result of a binary collision and moves in straight free-flight-path to the next collision locus.
      i. Under the RPA, the target is considered a gas of point-like atoms located at random locations. The collision partner is selected randomly, according to its relative fraction in

* Actually, a maximum number of layer and elements is specified at the beginning of file "corteo.c" but it can be increased, which requires recompiling the program.

the current layer.

ii. In TRIM, a free path value (exponentially distributed according to the RPA) would be computed using Eq. 3, involving the selection of a pseudo-random number and the computation of a logarithm. These operations requiring perhaps a hundred of processor cycles, it is more efficient to draw a free path value ℓ from a precomputed list stored in memory. Moreover, in order to compute the impact parameter and energy straggling, we will need $\sqrt{\ell}$, so at startup, Corteo creates a list of about 10^5 value of $\{\sqrt{\ln x_1}\}$ with x_1 uniformly distributed but randomly ordered between 0 and 1, and the flight length is computed as

$$\sqrt{\ell} = \sqrt{\ell_0}\{\sqrt{\ln x_1}\}, \ell = (\sqrt{\ell})^2 \qquad \text{Eq. (27)}$$

This procedure thus potentially saves the computation of billions of logarithms and square roots, but at the price of reusing several times the same values. Choosing list lengths that are prime numbers makes the cycling to occur only once every 10^{15} collisions in the current version of Corteo.

iii. A reduced impact parameter s is then randomly picked, in the case of TRIM using Eqs. 6 and 7. In order to avoid the computation of a square root and a division by ℓ, Corteo again precomputes a list of $\{\sqrt{x_2}\}$ with x_2 uniformly distributed but randomly ordered between 0 and 1, and a list of reciprocal values of $\{\sqrt{\ln x_1}\}$ is also stored in memory. The reduced impact parameter is obtained as

$$s = s_0\{\sqrt{x_2}\}\{1/\sqrt{\ln x_1}\} \qquad \text{Eq. (28)}$$

where $s_0 = 1/\sqrt{\pi a^2 \ell_0 N}$ is computed beforehand for each element in each layer. It is noted that since the lists of $\{\sqrt{\ln x_1}\}$, $\{1/\sqrt{\ln x_1}\}$ and $\{\sqrt{x_2}\}$ are randomly ordered, they are accessed sequentially during the simulation, allowing the processor to transfer several values in advance into its cache memory, thus making memory transfers efficient.

iv. Knowing the impact parameter and the energy, we can then compute the scattering angle. Using Eq. 14 would make it impossibly long (about a few ms each time on modern processors, billions of times). Biersack's `MAGIC` algorithm is much more efficient, but needs namely to compute the screening function (Eq. 1) perhaps several times so it still takes several hundreds of processor cycles. Moreover, we then still have to compute the angle components in the laboratory frame through lengthy Eqs. 16 or 18 and others. In order to avoid all that, Corteo reuses and improves the idea of Yuan *et al.*[14] who suggested to extract the values of $\sin^2\Theta_{CM}/2$ from a matrix indexed using the binary representation of floating point numbers. This method still requires a transformation to laboratory reference frame. We explain in Section 2.4 below that Corteo instead uses stored matrices of $\{\cos\theta\}$ and $\{\sin\theta\}$, the components in the laboratory reference frame.

v. The new flying direction of the atom is computed according to the rotation algorithm described in Appendix I and summarized by Eqs. 20. These equations require values of $\cos\omega$ and $\sin\omega$ for the azimuthal angle ω randomly selected between 0 and 2π. Again to avoid finding a random number and computing two trigonometric functions billions of times, Corteo relies on uniformly distributed but randomly ordered lists of $\{\cos\omega\}$ and $\{\sin\omega\}$ accessed sequentially.

c. The ion energy decreases as a result of nuclear and electronic energy loss.

i. For the nuclear energy loss, the energy transferred is given by Eq. 21, for which the

kinematic factor is precomputed, and the value of $\{\sin^2\Theta_{CM}/2\}$ is obtained from a matrix as for $\{\cos\theta\}$ and $\{\sin\theta\}$.

ii. The electronic energy loss is computed using Eq. 22, the stopping power coming from a table accessed using the binary representation of the energy as an index. Stopping power values for pure elements are currently obtained from SRIM's SRModule. Bragg's rule is assumed for compounds but a compound correction can be specified. Energy straggling in Eq. 22 is computed using Eq. 23. However, Corteo again uses precomputed (energy dependent) tables of $\{\Omega\sqrt{2}\}$ using the energy straggling models selected by the user. (In compounds, straggling is computed according to Bragg's rule.) A long list of $\{\mathrm{erf}^{-1}x_3\}$ values uniformly distributed in $x_3 \in]-1,1[$ is loaded from disk and randomly ordered when the program starts. Values from this list are used sequentially to compute Eq. 23.

d. The position of the ion is advanced to the next collision site following

$$[x', y', z'] = [x, y, z] + \ell[l', m', n'] \qquad \text{Eq. (29)}$$

The program checks if a layer boundary is crossed and readjust the flight length according to the density difference, and the energy loss is corrected for the distance crossed in the new layer. In the case of targets described by a bitmap, Corteo counts how many pixels were in two different materials, and adjust the stopping accordingly. However, if the ion crossed more than two materials during a single flight, Corteo only corrects for two of them and increments a counter about the number of times this occurrence.

e. The procedure starts back at step a. unless the ion reaches the bottom or the top of the target, or a certain depth as described in the next step. The process also stops if the particle has slowed down to a certain minimum energy set by the user. This minimum energy should be set relatively high (e.g. 1/10 of the beam energy), thus saving unnecessary computational efforts when the ion is down to an energy not useful for spectrum computation.[13]

This slowdown process (hereafter called Step 1) will also be applied to the outgoing ion as it leaves the material and crosses an eventual foil at the entrance of the detector. It is thus important to check if the trajectory simulation is correct, given the additional approximations made, namely in terms of discretization of the different contributions. A comparison between Corteo and SRIM 2006 simulations are found in Figs. 3-5. These figures compare the energy and directional vector components distributions resulting from ions slowdown process for different ion/substrate combinations. It is seen that Corteo's slowdown algorithm reproduces fairly well the distributions obtained with SRIM. In Corteo, the mean free path ℓ_0 was set to 1/100 of the layer thickness, thus involving the computation of 100 collisions on average for each ion. The simulation of 10^5 trajectories typically takes 1 s (plus a fixed time of about 2 seconds for program startup) compared to about 1000 s with SRIM.

2. Following Arstila,[13] when the incident atom reaches a specified depth (the program scans over depth), the *main collision* occurs: a recoil or a scattered particle is deliberately thrown within a relatively wide cone about the detector. The angular width of this cone is set by the user, and a procedure to find its value is described in Section 4.1.1. The components of the directional vectors of the incoming and outgoing ions at the *main collision* locus are stored in order to be able to compute the cross-section in the event that the outgoing ion is finally detected. Deliberately throwing the ions at the detector is the approximation that improves the simulation time by the largest factor (~10^4). Even if the cone within which the ions are thrown is wide (e.g. a half sphere), it makes the number of detected ions independent of the cross-section of that main collision, which is usually very small for backscattering or recoil events, these events thus

being usually extremely rare.

3. Step 1 is executed for the ion leaving the main collision site until the surface (or back) of the target is reached, or it reaches the minimum energy of the simulation.
4. If the outgoing ion reaches the correct surface (top of the sample, except for coincidence where it must be the bottom of the sample), a randomly selected offset is added to its position to account for the beam size on the surface. (The beam is assumed to be radially symmetric and to have a Gaussian section, so the spot on the surface is elliptical if the beam is tilted about the surface normal.) Note that in the case of a bitmap, such random offset is also applied, i.e. the purpose of a bitmap is merely to take into account microscopical geometrical effects such as inclusions or porosity, while the purpose of the offset applied at this step is to take into account macroscopic geometrical effects such as kinematic broadening.
5. The ion intersection with the detector is determined as described in Section 2.5 below.
    a. Detectors can be elliptical or rectangular (circular and square detector being special cases), and can be positioned and oriented arbitrarily in space. A detector can also be annular, but only a circular hole is allowed.
    b. In the case of coincidence, two detectors are defined and both must intersect either the scattered ion or the recoil trajectory to produce a detection event. They can be separate detectors or can overlap in order to represent both halves of a split annular detector.
    c. In the case of TOF methods, two detectors are defined, the first one being the timing foil and the second one being defined in the reference frame of the first.
    d. A layer can be defined in front of a detector, such as an absorber or a timing foil. In this case, the ion slowdown in this layer is computed using Step 1. This is useful to compute the straggling due to such foil, but it also may lead, for example, to a situation where an ion initially well-oriented towards the second detector of a TOF system is deflected away as it passes through the timing foil. Such effect, and its mass dependence, can be taken into account by MC simulations.
    e. Corteo also implements the virtual detector concept proposed by Arstila:[13] a detector wider than the actual detector is defined, and the trajectory of virtually detected ions is corrected to make them reach the real detector, including corrections to the kinematic factor during the main collision and a correction to the energy loss. Still, these corrections are made assuming that the trajectory is relatively straight and may lead to bad corrections when this is not the case, especially when one wants to simulate the effect of plural scattering. Corteo makes possible a virtual detector larger than the actual detector by different factors along its two axes. The correction can thus be made only in a direction where the trajectory length is not significantly changed (see Section 2.5.5).
6. The cross-section of detected ions is computed thanks to the components of the directional vectors of the incoming and outgoing ions stored at the time of the main collision. This requires the cross-section to be known for all possible angles. In addition to the Rutherford cross-section, Corteo implements the computation of Andersen's screening (valid for screening >85%), and can compute the cross-section using the Lenz-Jensen screening function (on which Andersen's screening is based) and or the ZBL (Universal) screening function. The User Interface (UI) can also interpolate the cross-section in a table. To this end, Corteo provides a "stamp" (text) with each event, indicating the incident ion isotope, the target atom isotope and the reaction product isotope, in order to help the UI figure out in which table to interpolate the cross-section, the table itself being provided by the user.
7. Corteo provides information about each detection event in the form of a list, which includes the:
    a. energy

b. cross-section of the main collision leading to this event,
c. normalization factor discussed in section 2.3,
d. energy of the incident ion just before the main collision,
e. cosine of the collision angle,
f. cosine of the angle the particles makes with the center of the detector at the exit of the main collision (which helps to define the width of the cone necessary to take account of multiple scattering),
g. Z of the target atom during the main collision,
h. layer (or material in case of bitmap) in which the collision occurred, and
i. stamp described in previous step.

Important point: When accumulating these events into a spectrum (as the UI does), the spectrum is incremented *not by one count, but by the probability of this event to occur*, that is, a factor proportional to the cross section $d\sigma/d\Omega$ of the main collision. This is further discussed in section 2.3 below.

8. The process starts back at Step 1 with a new incoming ion.

On processors featuring several cores, the job can be divided into the number of cores, making the simulation proportionally faster. This is done using the `pthread` library. Hence, it is efficient to ask a processor with N cores to run N threads, but the program is not intended for communications between processors. If one wants to run the program on several processors, a different instance of the program should be started on each with a different random seed. This can be done with an "array of jobs" in a PBS script (switch "-t"). Each instances of Corteo in the array of jobs can be initiated with a different random seed by passing $PBS_ARRAYID as an argument to Corteo.

2.3 Spectrum generated by Corteo

When accumulating from the event list the spectrum corresponding to the j^{th} layer, which has a density times a thickness $N_j t_j$, what has to be generated is actually the number of counts computed as

$$n_j(E) = N_j t_j \left(\frac{1}{i} \sum_i \frac{d\sigma_i}{d\Omega} \right)_E q \Delta\Omega , \qquad \text{Eq. (30)}$$

where $d\sigma_i/d\Omega$ is the cross-section of the i^{th} event, i the number of detected ions, q the experimental number of incident ions, and $\Delta\Omega$ the detector solid angle. However, i should actually be the number of detected ions if they were all detected. Some of them may not be detected because they reached the minimum energy, making i unreliable when simulating thick substrates. And when few of them are detected, i can feature a significant statistical fluctuation. i can instead be computed reliably as the number of ions that should have been detected, which is the number of incident ions I necessarily producing (in view of Step 2) a scattered or recoiled ion within a cone of solid angle $\Delta\Omega_{cone}$ times the fraction of the detector size compared to the cone at the detector, that is, $\Delta\Omega/\Delta\Omega_{cone}$. Therefore, $i = I\,\Delta\Omega/\Delta\Omega_{cone}$, leading to

$$n_j(E) = N_j t_j \left(\frac{1}{I} \sum_i \frac{d\sigma_i}{d\Omega} \right)_E q \Delta\Omega_{cone} . \qquad \text{Eq. (31)}$$

So in the event list returned by Corteo, one has $d\sigma_i/d\Omega$ and the normalization factor

$$N_j t_j \Delta\Omega_{cone}/I , \qquad \text{Eq. (32)}$$

and the product of these two factors needs only to be multiplied by q, the number of incident ions, in

order to get the sub-spectrum corresponding to layer j, and sum the sub-spectra $n_j(E)$ of the different layers in order to get the full spectrum. This is carried out automatically by the Corteo User Interface (CorteoUI), described in Chapter Error: Reference source not found, and allows for a correction of the cross-section in post-processing.

2.4 Scattering matrix and indexing mechanism

As mentioned in Step 1.b.iv, during ions slowdown computation, Corteo gets the scattering angle components ($\cos\theta$ and $\sin\theta$) from a matrix indexed using the binary representation of floating point numbers. A similar idea was proposed by Yuan *et al*.[14] and allows the matrix to be logarithmically distributed without having to compute logarithms, the index being simply obtained by a type change (from floating point to integer) and subtracting an offset. However, the indexing mechanism proposed by Yuan *et al*. was linear in terms of impact parameter, and suffered from imprecisions at high energy and small impact parameters (the type of collision often involved in plural scattering). In addition, their implementation returns $\sin^2\Theta_{CM}/2$ so a conversion of the scattering angle components in the laboratory frame through lengthy Eqs. 16 or 18 and others is still required, involving the computation of several square roots and an inverse trigonometric function. Their implementation also included an interpolation that involved other operations including divisions. Still, they were able to achieve this with very limited memory resources compared to today's standards, processors now featuring much more cash memory (several Mb) than the total memory of a desktop computer at the time (640 Kb).

In Corteo, the method is improved by: i) using and indexing mechanism based on binary representation of both the energy and the impact parameters, and ii) by computing fine-grained matrices of $\{\cos\theta\}$ and $\{\sin\theta\}$, therefore bypassing the conversion to laboratory reference frame and interpolation. Operations other than multiplication, addition and bit-shifting are avoided during the computation of ions slowdown (Steps 1.b-d.). A few divisions are still involved and represent the lengthiest operations of the process.

This, however, is done at the price of almost random access to a few megabytes of data stored in memory. But modern CPUs often feature several megabytes of cache memory. Plus, the access is not fully random because the ion energy decreases continuously between collisions. If the energy change due to MS is small (and it usually is), the index value for the reduced energy may remain about the same for a few collisions, while the impact parameter is accessed truly randomly from one collision to the other. Since a processor usually transfers from the main memory to its cache memory a sequential batch of data including more than the required data, the matrices are stored energy-wise, that is, data corresponding to the different impact parameter of a certain energy are stored consecutively. Consequently, the batch transfer includes several values related to the randomly selected impact parameter at the same energy. Taking care of such issue accelerates the program by about 10% over storing matrices “impact-parameter-wise”.

The principle for indexing the matrix thus relies on the digital representation of floating point numbers in a computer. Index determination for a given floating point value assumes:

- 32-bit long integers and single precision floating point values
- IEEE Standard 754 representation for single precision floating point values:
    - bit 31: sign
    - bit 30-23: exponent (base 2), biased by 127 (i.e. value of exponent for 2^0 is 127)
    - bit 22-0: mantissa (i.e. fraction of 1, base 2)

For example, the number $1.23456\text{x}10^7$ is represented as

$$\underbrace{0}_{\text{sign}}\ \underbrace{10010110}_{\text{exponent base 2}}\ \underbrace{01111000110000100000000}_{\text{mantissa}}$$

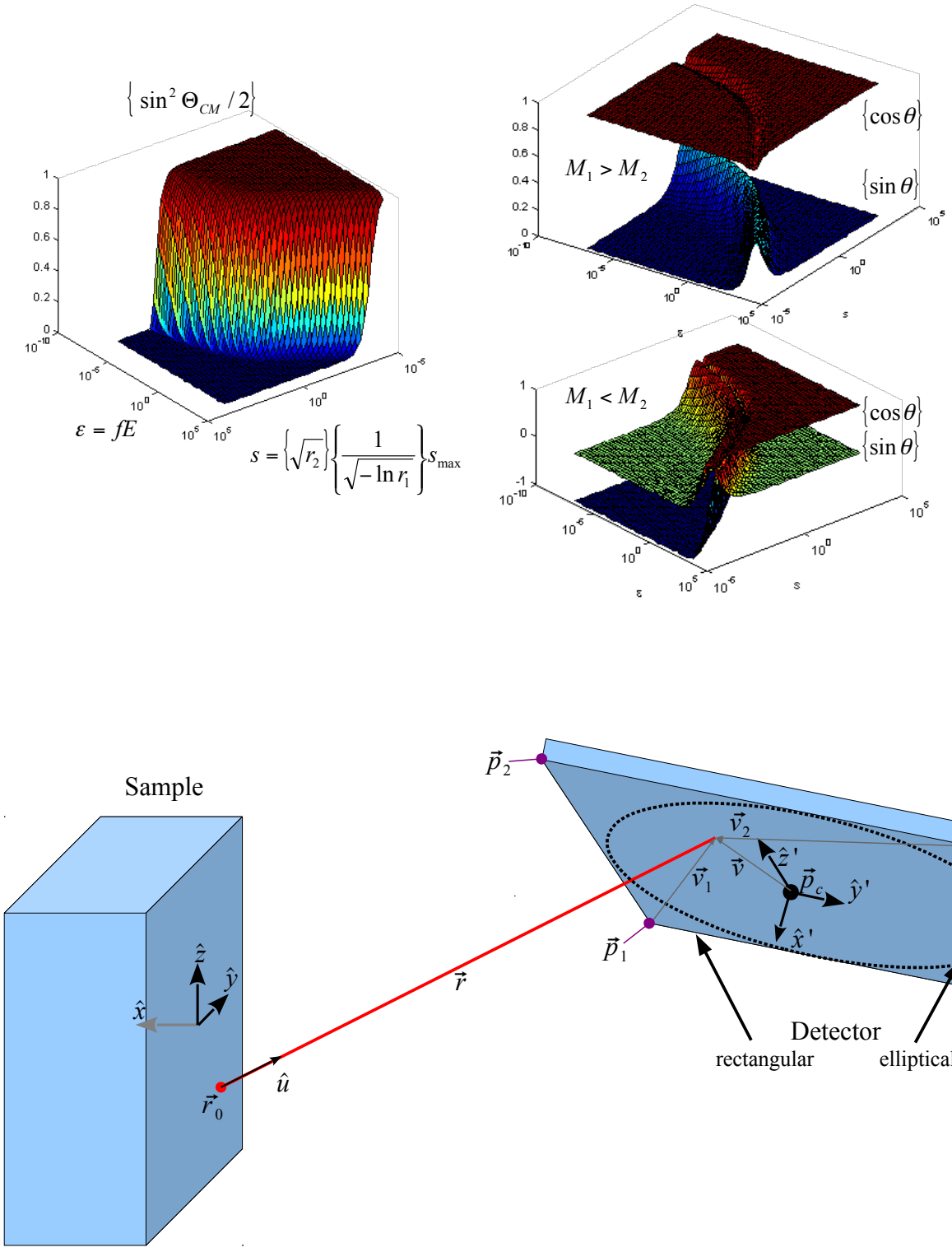


Fig. 10: Shematic representation of the vectors associated with detector definition and trajectory intersection

This can be architecture dependent, but is the convention used by Intel and AMD for laptop and desktop computer processors, therefore covering most personal computers. Be careful if you intend to port the program to other architectures.

If one uses only the exponent bits of a given floating point number, this will return an integer index that increases by 1 for each power of 2 of the floating point value. In order to achieve a more refined indexing, one can use a few mantissa bits. Using one bit will divide in 2 the interval between two successive powers of 2, 2 bits divides in 4, 3 bits in 8, 4 bits in 16, etc. Using 6 mantissa bits thus corresponds to a round-off error of ±0.8% on a given input value and was chosen for the program. It results in an error of less than 2% on values of $\{\sin^2\Theta_{CM}/2\}$, that is, on the exact flight direction after a single collision. After several collisions or over several ions, this error should smear out and becomes negligible.

The index is thus simply the integer value of the 8 exponent bits (to which we subtract 127 because of the bias) plus the first 6 mantissa bits, turned into an integer thanks to some C bit-shifting trickery. The program sets a minimum input value for which the index is 0. In order to get an index, one would then have to divide an input value by this minimum to get the corresponding index. It turns out that if the minimum value is selected to be a power of 2, the division corresponds to a simple subtraction to the index that can be carried out at the same time as the exponent bias correction. Here is how the function that returns an index for a reduced energy looks like :

```
unsigned long Eindex(float val) {
  unsigned long ll;
  ll = *(unsigned long *)&val; // not fully legal, use -no-strict-aliasing switch in gcc
  ll = (ll >> SHIFTE)-BIASE;
  return ll;
}
```

where, considering a minimum reduced energy of $2^{-19} \approx 1.9\text{x}10^{-6}$, the code includes the following

```
#define BIASE 6912   // (127-19)*2^6: exp. bias corr. while performing division by 2^(-19)
#define SHIFTE 17    // keep 6 of the 23 mantissa bits (23-6=17)
```

The actual code also makes sure that the returned index is smaller than matrix dimension, at a cost of about 10% in computation speed. Here are examples of the index returned for different floating point values:

Value	Value/minimum (2^{19})	Index
0.0000019073486328	1	0
0.0000019371507278	1.015624881	0
0.0000019371509552	1.015625	1
0.0000019669530502	1.031249881	1
0.0000019669532776	1.03125	2
0.0000038146970383	1.999999881	63
0.0000038146972656	2	64
0.0000038719176700	2.029999971	64
0.0000171661376953	9	200

When the program setup is ran, a table of $\{\sin^2\Theta_{CM}/2\}$ is computed once and for all* using the Gauss-Mehler quadrature (Eq. 14), for all pairs of reduced energies ε and reduced impact parameters s, determined by the floating point values corresponding to each index. (5.2 million values are computed, and accessed billions of times in simulations.)

Minimum values of ε and s (index=0) are set to a fixed values (hard coded in the bias value). Then, when the program starts, among all the tables computed, it produces matrices of $\{\cos\theta\}$ and $\{\sin\theta\}$ using Eqs. 17 and 18 for *all pairs of projectile/target element masses* involved in the slowdown computation. In the current Corteo version, this is 2x21 Mb per distinct element present in layers description for RBS, and another factor of 2 when carrying out ERD simulations since the recoil is not the same ion as the incident ion. For this reason, layers description should be kept to a minimum, avoiding trace elements or isotopes. We have to keep in mind that these matrices are used only for computing the projectiles slowdown, and not the "main collision" for which the target element is specified independently of the target composition. The layer composition as a slowdown medium and in order to simulate spectrum components is specified separately. It is then possible to simulate each isotopes as a main collision partner, but specify only a single, average isotope for slowdown computation purposes, and save a lot of memory burden. This is what happens when someone checks the "Simulate isotopes" for an element in the User Interface.

Figure 9 shows how the matrices look like when represented on logarithmic axes of reduced energy and impact parameters. When $M_1 < M_2$, $\cos\theta$ goes from -1 (180º backscattering) at small energies and small impact parameters, to +1 (collision missed) at high energies and large impact parameters. For $M_1 > M_2$, $\cos\theta$ rather stay positive since a heavy projectile will continue forward after colliding with a lighter target atom. The gap between $\{\cos\theta\}$ and $\{\sin\theta\}$ reflects the critical angle.

On point to retain from all this discussion is that using such tables implies a trade-off on *precision* and *memory access*. Still, if the total table size is limited, memory access is much faster than computing trigonometric or transcendental functions (especially if the useful parts of the matrices is small enough to fit in the processor cache memory), and because ion slowdown is a stochastic process, the imprecision brought by this method should smear out over several collision or several ions simulated differently.

2.5 Detection

Corteo implements rectangular and elliptical detectors that can be positioned and oriented arbitrarily in space, rather than just a circular detector with its center sitting in the $z = 0$ plane and its normal oriented towards target origin. This section somewhat details the theory behind detection computation. Basically, it comes down to determine if a line intersects a rectangle or an ellipse in three dimensions, with an attempt to make the process computationally efficient while not giving up on generality, such as the possibility to have arbitrarily positioned and oriented detectors.

In Corteo, a detector is defined by three points in space defining three corners of a rectangle, as we will see below. A circular detector would be more naturally defined by its central point, a normal vector to its surface and a radius. However, we often need to transform ion trajectories in the reference frame of the detector in order, for example, to compute ion slowdown through its foil or to simulate its trajectory in a time-of-flight camera. Another important feature we want to include is the implementation of elliptical virtual detectors, that is, a circular detector elongated in the direction parallel to the target surface but not in the other direction. This can make the correction of awkward trajectory less problematic as discussed below. In order to be able to specify an elongation in the right

* The computation assumes ZBL screening, and the table has to be recomputed if one wants to use a different potential.

direction, it is thus necessary to specify properly a reference frame for detectors, even for circular ones.

For these reasons, detectors are all defined by specifying three points in the sample reference frame, $\vec{p}_1, \vec{p}_2, \vec{p}_3$, that define a rectangle. Detection in elliptical and rectangular detectors differ only by a slightly different condition described below, the other operations being the same. (Circles and squares are special cases of ellipses and rectangles, respectively.) $\vec{p}_1, \vec{p}_2, \vec{p}_3$ are described by 9 coordinates specified by the user, while strictly speaking, only 8 are required to define a rectangle in space. But other conventions would have only made detector definition more difficult to understand. Corteo only checks that the sides of the rectangles are perpendicular, that is, $(\vec{p}_2-\vec{p}_1)\cdot(\vec{p}_3-\vec{p}_2)\simeq 0$. The program also checks that detectors are defined in such a way that their surface normal, defined as $(\vec{p}_3-\vec{p}_2)\times(\vec{p}_2-\vec{p}_1)$, points towards the target plane, as discussed below.

2.5.1 Intersection

The different reference frames, points, vectors, planes and trajectories discussed in this section are illustrated in Fig. 10. The trajectory of an ion emerging from the sample surface is given, in the sample reference frame, by

$$\vec{r}=\vec{r}_0+t\,\hat{u}\,,\quad t\geq 0\,. \qquad \text{Eq. (33)}$$

where $\vec{r}_0$ and $\hat{u}$ are the point at which the eventually detected ion emerges from the surface and its directional vector, respectively. We now want to compute the intersection of this trajectory with a detector. Let first define a reference frame for the detector. Two of the axes are defined as unit vectors parallel to the sides of the rectangle defined by $\vec{p}_1, \vec{p}_2, \vec{p}_3$:

$$\begin{aligned}\hat{z}'&=(\vec{p}_2-\vec{p}_1)/|\vec{p}_2-\vec{p}_1|\,,\\ \hat{y}'&=(\vec{p}_3-\vec{p}_2)/|\vec{p}_3-\vec{p}_2|\,,\\ \hat{x}'&=\hat{y}'\times\hat{z}'\,.\end{aligned} \qquad \text{Eqs. (34)}$$

Consequently, $\hat{x}'$ is a unit vector normal to the detector plane.

The detector plane being defined as $x'=0$, that is, in the y', z' plane of the detector reference frame, the detector plane is thus represented in the sample reference frame by the equation

$$\hat{x}_x' x+\hat{x}_y' y+\hat{x}_z' z+d=0 \qquad \text{Eq. (35)}$$

where the $\hat{x}_i'$ are the components of $\hat{x}'$ in the sample reference frame and $d=-\hat{x}'\cdot\vec{p}_1$. It will also be useful to define the center of the detector:

$$\vec{p}_c=(\vec{p}_3+\vec{p}_1)/2\,. \qquad \text{Eq. (36)}$$

The ion trajectory, Eq. 33, intersects the detector plane when

$$\hat{x}_x'(r_{0,x}+tl)+\hat{x}_y'(r_{0,y}+tm)+\hat{x}_z'(r_{0,z}+tn)+d=0 \qquad \text{Eq. (37)}$$

or, solving for *t*, when

$$t=t_0=-\frac{d+\hat{x}'\cdot\vec{r}_0}{\hat{x}'\cdot\hat{u}}\geq 0 \qquad \text{Eq. (38)}$$

If $t_0 < 0$, the ion is heading away from the plane and is rejected. For $t_0 > 0$, the intersection point in the sample reference frame is

$$\vec{r}=\vec{r}_0+t_0\hat{u}\,. \qquad \text{Eq. (39)}$$

2.5.2 Orientation of the detector

A detector has to be oriented and positioned so that trajectories intersect its surface from the external side, that is, the side of the surface normal. This is necessary to properly carry out the transformation of ion trajectory into the detector reference frame when considering an absorber or the first timing foil of a TOF camera. The program achieves that by checking if a trajectory starting from the center or the

detector $\vec{p}_c$ and pointing in the $\hat{x}'$ direction intersect the sample surface $x = 0$. This is the case if the solution for t' in

$$(\vec{p}_c + t'\hat{x}')_x = 0 \qquad \text{Eq. (40)}$$

is positive, i.e. if $\vec{p_{c,x}}/x'_x \leq 0$. This imposes the detector to be in front of the target and having $\hat{x}'$ in the same orientation as $\hat{x}$ (i.e. $\hat{x}\cdot\hat{x}' > 0$) , or to have the detector behind the target (as for transmission, coincidence) and having $\hat{x}'$ in an orientation opposite to $\hat{x}$ (i.e. $\hat{x}\cdot\hat{x}' < 0$) . In both cases, the detector normal vector somehow faces the scattering/recoiling events.

2.5.3 Rectangular detector

Let now find if the intersection point on the detector plane is within the rectangle defined by $\vec{p}_1, \vec{p}_2, \vec{p}_3$. Defining

$$\vec{v}_1 = \vec{r}_0 + t_0\hat{u} - \vec{p}_1 ,$$

a vector relating $\vec{p}_1$ and the impact point, and Eqs. (41)

$$\vec{v}_2 = \vec{r}_0 + t_0\hat{u} - \vec{p}_3 ,$$

a vector relating $\vec{p}_3$ and the impact point, the particle is within the rectangle if the four following conditions are met:

$$\begin{aligned} &\hat{z}'\cdot\vec{v}_1 \geq 0 \ \text{(above bottom)} \\ &\hat{z}'\cdot\vec{v}_2 \leq 0 \ \text{(below top)} \qquad \text{Eqs. (42)} \\ &\hat{y}'\cdot\vec{v}_1 \geq 0 \ \text{(right of the left side)} \\ &\hat{y}'\cdot\vec{v}_2 \leq 0 \ \text{(left of the right side)} \end{aligned}$$

This method should be relatively efficient because it leaves only $t_0, \vec{v}_1, \vec{v}_2$ and four dot products to compute during the simulation, with only one division for t_0.

2.5.4 Elliptical detector

In the case of elliptical detectors, to find if the impact point is within the ellipse illustrated in Fig. 10, we define

$$\vec{v} = \vec{r}_0 + t_0\hat{u} - \vec{p}_c , \qquad \text{Eq. (43)}$$

a vector between the detector center and the intersection point. The particle is within the ellipse included in the rectangle defined by $\vec{p}_1, \vec{p}_2, \vec{p}_3$ if

$$\frac{(\vec{v}\cdot\hat{y}')^2}{a^2} + \frac{(\vec{v}\cdot\hat{z}')^2}{b^2} \leq 1 \qquad \text{Eq. (44)}$$

where $a = |\vec{p}_3 - \vec{p}_2|/2$ and $b = |\vec{p}_2 - \vec{p}_1|/2$ the half-length of the two axes of the ellipse. This method leaves $t_0, \vec{v}$ and two dot products to be computed during the simulation, with again only one division for t_0 provided that $1/a^2$ and $1/b^2$ are computed in advance. If an annular detector is considered, the particle is detected if $\vec{v}^2 \geq r^2_{\text{annulus}}$.

2.5.5 Virtual detector

Arstila proposed the concept of a virtual detector[13], that is, a detector which is e.g. 10 times larger in diameter than the actual detector, making simulation 100 times more efficient. Here, this approximation is implemented by letting the user decide if he/she wants the detector to be enlarged by different factor in each direction. If an ion is detected by the virtual detector, the intersection point is scaled back to the size of the actual detector. This requires a correction (rotation) to the directional vector at the exit of the

main collision, $\hat{u}_c$, so we can compute the collision cross-section and kinematic factor according to this corrected vector. Let $\vec{r}_v$ be the spot where the particle hits the virtual detector. Given H and W the factors by which the height ($\hat{z}'$ direction) and the width ($\hat{y}'$ direction) of the detector have been multiplied, the spot where the event is rescaled† on the real detector is

$$\vec{r}_r = \frac{(\vec{r}_v - \vec{p}_c)\cdot\hat{y}'}{W}\hat{y}' + \frac{(\vec{r}_v - \vec{p}_c)\cdot\hat{z}'}{H}\hat{z}' + \vec{p}_c \,. \qquad \text{Eq. (45)}$$

We want to find the rotation matrix that rotates $\vec{r}_v$ to $\vec{r}_r$ so we can rotate $\hat{u}_c$ the same way. (Note that $\hat{u}_c$ does not point in the same direction as $\vec{r}_v$ because of multiple collisions, although it is generally close, but not always as in the case of awkward trajectories.) The rotation formula relating $\vec{r}_v$ to $\vec{r}_r$ is

$$\vec{r}_r = \vec{r}_v\cos\theta + \hat{c}\,(\hat{c}\cdot\vec{r}_v)(1-\cos\theta) + (\vec{r}_v\times\hat{c})\sin\theta \qquad \text{Eq. (46)}$$

where $\hat{c} = \vec{r}_v\times\vec{r}_r / |\vec{r}_v\times\vec{r}_r|$ is a unit vector normal to the $\vec{r}_v$, $\vec{r}_r$ plane, and θ is the angle between $\vec{r}_v$ and $\vec{r}_r$. Because $\hat{c}$ is perpendicular to $\vec{r}_v$, the second term cancels out and the formula reduces to

$$\vec{r}_r = \vec{r}_v\cos\theta + (\vec{r}_v\times\hat{c})\sin\theta \qquad \text{Eq. (47)}$$

The rotation matrix bringing $\vec{r}_v$ to $\vec{r}_r$ is thus

$$R = \begin{pmatrix} \cos\theta & c_z\sin\theta & -c_y\sin\theta \\ -c_z\sin\theta & \cos\theta & c_x\sin\theta \\ c_y\sin\theta & -c_x\sin\theta & \cos\theta \end{pmatrix}, \qquad \text{Eq. (48)}$$

where the c_i are the components of $\hat{c}$. Therefore, the components of $\hat{u}_c' = R\,\hat{u}_c$ (the corrected directional vector at the exit of the main collision) are

$$\begin{aligned} l_c' &= l_c\cos\theta + (m_c c_z - n_c c_y)\sin\theta \,, \\ m_c' &= m_c\cos\theta + (n_c c_x - l_c c_z)\sin\theta \,, \\ n_c' &= n_c\cos\theta + (l_c c_y - m_c c_x)\sin\theta \,. \end{aligned} \qquad \text{Eqs. (49)}$$

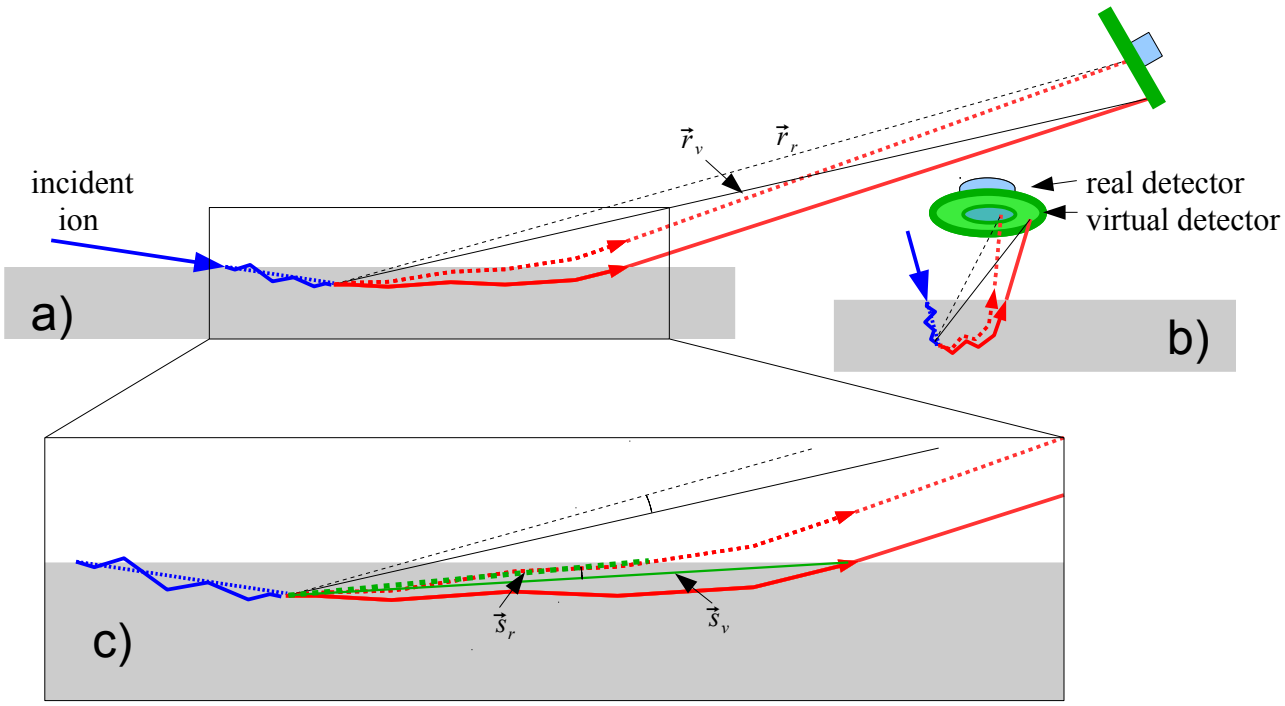
The whole process namely involves three square root evaluations, but is does not impair significantly the execution speed because the calculation is carried out only on the detected ions which usually represent a small fraction of the number of incident ions.

The correction to the kinematic factor is usually the most significant. However, it is important to note that the correction carried out this way assumes that the trajectory is relatively straight. This results in some bad side effects on awkward trajectories. Fig. 11 illustrates the effect of the correction applied to such case. In Fig. 11a), it is shown that an up-down rotation of the trajectory results in a significant difference in distance traveled by the ion into the material, while in Fig. 11b) it is seen the distance traveled after a left-right correction is not very different. In the rotated trajectory of Fig. 11a) (dotted red line) the portion that was initially in the sample now extends significantly outside the sample. One could not just scale down the length of the trajectory because it would make each of the collisions resulting in a larger angle than the actual process, or more numerous collision than what would have actually occurred if this corrected trajectory was followed.

Still, we need to correct the energy losses for the difference of distance traveled in the sample. The way the correction is carried out in Corteo is to scale the energy loss of the outgoing ion by a factor $|\vec{s}_r|/|\vec{s}_v|$ where $\vec{s}_v$ is a vector relating the main collision locus to the actual exit point (solid green line in Fig. 11c) and $\vec{s}_r$ is a vector parallel to $R\,\vec{s}_v$ and relating the main collision locus to the surface (represented by the dotted green line in Fig. 11c). In the case of strait line trajectories, vectors $\vec{s}_v$ and $\vec{s}_r$ would be co-linear to $\vec{r}_v$ and $\vec{r}_r$ respectively.

The correction applied this way is perfect for strait line trajectories, but looses its perfection as the

† In Arstila's implantation, a spot $\vec{r}_r$ is selected randomly on the real detector. Applying a scaling instead allows us to avoid the up-down correction to the trajectories discussed below and only consider left-right corrections.



On the other hand, the correction to the energy loss is small when applying left-right corrections to the trajectories. In Fig. 11b), the vectors $\vec{s}_v$ and $\vec{s}_r$, while badly representing the solid and dotted red lines, respectively, would anyway have about the same length, $|\vec{s}_r|/|\vec{s}_v|$ therefore being close to 1 as for the ratio of the length of the actual and corrected trajectories. For this reason, Corteo provides the possibility to enlarge the detector by different factors along its two axes. One can therefore make the virtual detector larger only in the direction parallel to the surface, and rotating the trajectories from left to right will only involve a small correction to the distance traveled into the material, hence a small correction to the stopping, while the corrections to the kinematic factor and cross section may be significant because of the change in the angle of the main collision.

2.5.6 Inside the detector

Once a particle is found to reach (or made to reach, after the virtual detector correction) the detector, we may need to compute its slowdown through a foil. For instance, in a TOF system, it not only causes more energy loss and energy straggling, but it may also lead to particle deflections at different angles for different masses, an effect that a user may want to consider. The absorber used in ERD can be specified. Dead layer effects is solid-state detectors could also be taken into account.

For this part of the calculation, the particle trajectory is transformed to the reference frame of the detection system $\hat{x}', \hat{y}', \hat{z}'$. The components of the directional vector $\hat{u}$ in this new reference frame can be obtained by the projection of the vector on the new axes, that is

$$\hat{u}' = l'\hat{x}' + m'\hat{y}' + n'\hat{z}' \qquad \text{Eq. (50)}$$

where

$$\begin{aligned} l' &= \hat{u}\cdot\hat{x}' = l\,x_x' + m\,x_y' + n\,x_z' \\ m' &= \hat{u}\cdot\hat{y}' = l\,y_x' + m\,y_y' + n\,y_z' \\ n' &= \hat{u}\cdot\hat{z}' = l\,z_x' + m\,z_y' + n\,z_z' \end{aligned} \qquad \text{Eqs. (51)}$$

where x_i', y_i' and z_i' are the components of $\hat{x}'$, $\hat{y}'$ and $\hat{z}'$ in the sample reference frame. The impact point in the detector reference frame is then

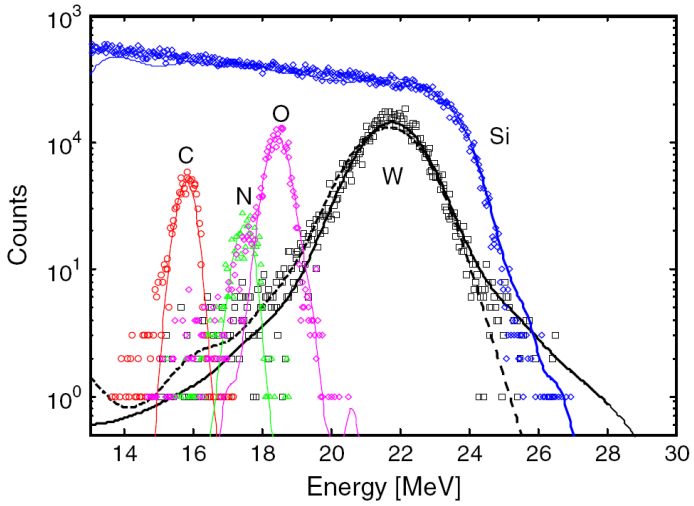
$$\vec{r}_0' = 0\,\hat{x}' + (\vec{v}\cdot\vec{y}')\,\vec{y}' + (\vec{v}\cdot\vec{z}')\,\vec{z}' \qquad \text{Eqs. (52)}$$

since the particle hits the detector plane at $x' = 0$. ($\vec{v}$ was defined in Eq. 44.) The trajectory in the new reference frame is thus

$$\vec{r}' = \vec{r}_0' + t'\hat{u}'$$

where $t' = 0$ at the impact point.

Slowdown through a foil can then be computed using Steps 1.b-d. In the case of a TOF detector, if the ion has emerged in the right direction, its intersection with a second detector is computed. This second detector is also defined by three points, but *with its coordinates specified in the reference frame of the first detector*.



3 Corteo input and output files

Corteo itself is a program without user interface, programmed in plain C. As it does not rely on any non-standard C library, it can in principle be compiled with any C compiler on any platform. One exception to that is the library used for multi-threading, which is implemented with the POSIX `pthread` interface. (On multi-core processors, it is necessary to run a number of threads that is a multiple of the number of cores in order to benefit from the full computing capability of the processor. The number of threads is specified by the user.) The program is distributed under the terms of the Gnu General Public License (GPL)[1] and can be downloaded for free and modified, provided that redistribution of modified version of the program is made easily available with the source code to anyone.

The program is an executable that can be called from any other program, namely a scripting languages such as Python or Matlab, in order to batch-process simulations or carry out optimizations. Most users will probably use the Corteo User Interface described in the next chapter, but power-users in need for special features unavailable from the user interface will have to switch to the more flexible call from a scripting language.

Corteo can be called with 0, 1, or 2 parameters. Called without parameters, it will read all input values from a file named `corteo.in`, described below. If the target is described by a bitmap, it is read from `corteo.bitmap`, which format is also described below.

If one parameter is specified, it has to be an integer number that is used to seed the random number generator. This number overrides the one specified in `corteo.in`. Hence, several instances of the same simulation can be started on a cluster computer, one per processor, each with a different random seed.

If a second parameter is specified, the first still has to be a random seed, and the second has to be a

filename without extension (but it may include a path). In that case, the input will be read from that filename to which the extension `.in` is added, samething for the `.bitmap` file, and the output that would normally be written to `corteo.detect` and other files, will be written to that filename to which the extension `.detect` or others is added.

3.1 Corteo input file

In Corteo, lengths are in Å and energies in eV. Here is an example of `corteo.in` file that will be described line-by-line.

```
% technique, mean free path, cone angle [deg], minimum energy [eV], energy straggling, writeDetails,
% screening, cutoff angle, random seed, nthreads
1 50 90 9e4 3 0 1 20 8550983 8
% beam Z, atomic mass, number of ions, beam energy [eV], beam radius [A]
2 4.002603254 100000 1.5e+06 15000
% incident beam directional cosines
0.707107 0 0.707107
% number of layers, number of traveling atoms
3 1
% layer thickness, roughness, atomic density, number of elements, number of projectiles
400 0 0.04977 1 1000000
% Z, mass, proportion, Edisp, Ebind, simNaturalAbundance,
% N isotopes to simulate, N*[fraction, mass, use CS table]
 14 0 1 20 2 1 3 0.922297 27.97692653 0 0.046832 28.9764947 0 0.030872 29.97377017 0
% layer thickness, roughness, atomic density, number of elements, number of projectiles
500 0 0.05525 1 1000000
% Z, mass, proportion, Edisp, Ebind, simNaturalAbundance,
% N isotopes to simulate, N*[fraction, mass, use CS table]
 73 0 1 20 2 1 2 0.00012 179.9475457 0 0.99988 180.9479958 0
% layer thickness, roughness, atomic density, number of elements, number of projectiles
2e4 10 0.04977 1 1000000
% Z, mass, proportion, Edisp, Ebind, simNaturalAbundance,
% N isotopes to simulate, N*[fraction, mass, use CS table]
 14 0 0.3636 20 2 1 3 0.922297 27.97692653 0 0.046832 28.9764947 0 0.030872 29.97377017 0
0 0 0 1 1 1
 1 1 1 1 1
% detectors: isEllipse p1x p1y p1z p2x p2y p2z p3x p3y p3z r_annulus
0  -4.34e+08 1.90e+08 -4.33e+08  -4.34e+08 1.45e+08 -4.50e+08  -4.41e+08 1.42e+08 -4.43e+08  0
0  -1 0 0  -1 1 0  -1 1 -1  0
% virtual detector enlargement factors: width, height
1 1
% ion Z, and its compound correction in each layer
2 1 1 1
```

First, it is seen that the user is allowed to put comment lines. Any line beginning by something else than a numerical character or a space is interpreted as a comment. On the other hand, any line beginning with a numerical character or a space must contain the expected number of parameters of the right type. (Glitches could occur if a negative number is specified for an unsigned value or a floating point for an integer.) However, more characters that the required number of parameters can be included at the end of a line, they will just be ignored. The "%" symbol used here is just an arbitrary choice to outline that these lines are comments. (In the text that follows, comment lines are not taken into account when referring to e.g. "the next line".)

3.1.1 Simulation-specific parameters

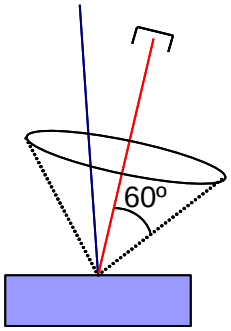
The first data line, in the example above "`1 50 90 9e4 3 0 1 20 8550983 8`", contains general parameters that control the simulation. The first parameter is the "technique". Possible values are:

- 1: RBS
- 2: RBS-TOF (not fully tested)

- 3: ERD
- 4: ERD-TOF
- 5: two-detector coincidence (identical ions only)
- -1: implantation (tests underway, works pretty well except for hydrogen and some other cases)
- -2: full cascade (tests underway, known not to predict the right number of vacancies)
- -3: full RBS: same as implantation/irradiation, but with with detection of out-comming ions
- -4: produces an energy-depth map based on implantation in the form of a matrix
- -5: produces a list of energy-depth

The "full RBS" simulation is indeed as if one would run SRIM until a sufficient number of incident ions reach the detector. The interest is that the main-collision approximation is avoided. However, it requires the simulation of 1 to 100 billion incident ions, typically requiring hundreds of core-hours, or a few hours on a 100-core machine.

The second parameter is the mean free path ℓ_0 used to compute Eq. 27. It indirectly selects the number of collisions each ion will suffer. Users must be warned that this is a quite dangerous parameter to play with. Care has to be taken to make the solution independent of this parameter. If one enters 0 for this value, ℓ_0 will be computed with Eq. 4. However, this value is usually unnecessarily small and will make simulations very long. A hidden feature is that one can enter here a negative number. This will tell Corteo to leave ions trajectory undeflected. Only the energy of each collision is subtracted. Hence, it allows simulations without multiple scattering (MS), only its contribution to energy straggling. This is useful in order to outline the effect of MS on a spectrum or to compare with other simulation programs.



The third parameter of the first parameter line is the cone angle, expressed in degrees. This sets the width of the cone around the detector axis in which the scattered ions or recoils are thrown, as described in Step 2 of Section 2.2 and illustrated in Fig. 13. Setting a wide cone will allow to better account for awkward trajectories (i.e. ions initially directed away from detector but scattered back to the detector by MS) but will also result in a smaller fraction of detected ions. The cone angle should be wide enough to include most (e.g. 99%) of the MS effects. In order to estimate this angle, one one can run a simulation for which the cone angle is set to a large value, and look at the distribution of the angle the particles leaving the main collision locus make with the detector, for events that did reach the detector. The info required to compute this distribution is provided in the Corteo output (see below) and the user interface can produce such a graph, as described in Section 4.1.1.

The next parameter is the minimum energy to which ions are followed. It would be useless and very long to follow each ion until they stop if one is interested only in the part of the spectrum above a certain energy. Setting the minimum energy to a relatively high value (e.g. 1/5 of the beam energy) can save considerable computational efforts. The simulated spectrum below this energy won't be valid.

The fifth parameter defines the type of energy straggling considered. Possible values are:

0. No energy straggling
1. Bohr straggling (Eq. 24)
2. Chu correction (Eq. 25)
3. Yang *et al.* model (Eq. 26)

Then, if the "write details" is non-zero, a lot more information will be written to the output file, corteo.detect, for each event, such as info about main collision and detection, and in the case of full RBS simulations, the few largest values of scattering angle and their and corresponding depth. Watch

out! It spits a lot, and could generate Gb of data.

The next parameter indicates which screening function should be used to correct the cross-section. Possible values are:

0) use plain Rutherford cross-section
1) use Andersen screening
2) compute the cross-section using consider the Universal potential
3) compute the cross-section using the Lenz-Jensen screening function,
4) interpolate in a table (or actually, provide a "stamp" indicating in which table to interpolate)

The next parameter in this series is a cut-off angle used when simulating RBS spectra. If the cone angle is set to a very wide value (almost 180°), this results in main collisions possible at very small scattering angle, thus with a diverging cross section. If such low angle scattered ion is deflected back to the detector by MS, it will contribute an enormous peak in the spectrum. This is the main drawback of the "main collision" approximation. It is usually not a problem when considering backscattering (i.e. RBS with a scattering angle larger than 90º), but it becomes one when considering forward scattered particles, e.g., in ERD. This parameter can be used to filter out these events, and their contribution can be estimated using a Full RBS simulation where only the trajectories with deflections below this angle are considered. See Ref. [15] for details. ERD simulations do not suffer from this problem.

3.1.2 Beam description

The next series of parameters, "`2 4.002603254 100000 2e+06 5e+07`" in the example above, describes the beam (except its direction) and the number of incident ions to simulate. The parameters are the incident ions atomic number Z_1, their mass M_1 (amu), the number of ions to simulate I, the beam energy E_0 (eV) and the beam radius r (Å). If M_1 is set to 0, the mass of the most abundant isotope will be considered. In the current Corteo version, the beam is considered to have a Gaussian distribution and to be radially symmetric. *The beam radius actually specifies its standard deviation*. However, its correct projection on target surface is taken into account, the spot size being larger along the tilt direction of the beam.

The subsequent three parameters ("`0.707107 0 0.707107`") are the components of a unit vector in the direction of the beam (also called the directional cosines) in the sample reference frame where $\hat{x}$ is pointing inside the target and $\hat{y}, \hat{z}$ are in the surface plane. The vector defined here need not to be normalized to 1, normalization is carried out automatically at program start. In the current example, the beam hits the sample at 45º in the y=0 plane.

3.1.3 Target description

The next line of parameters ("`3 1`" in the example), specify the number of layers in the simulation, and the number of different ions for which a trajectory is followed. That second number is required in order to read, further below in the file, the right number of compound correction parameters.

If the target is represented as a bitmap, the fact that a bitmap is used is specified by indicating a negative number as the number of layers. The absolute value of the is number indicates the number of materials/colors in the bitmap (excluding vacuum). In that case, the parameter line additionnally includes the dimensions in x, y, and z of the bitmap, whether or not a substrate layer sits under the bitmap, and the number of pixels/Å. An example of that parameter line when the target is described by a 2D bitmap 512 pixels thick and 230 pixels wide without underlying substrate, and 0.2 pixels/Å (i.e. each pixel is 5 Å in size) would be "`-3 1 512 230 1 0 0.2`", since a 2 D bitmap dimension in z is 1.

The maximum number of layers or materials/colors is hard-coded and currently set to 20 but this can

be changed easily, requiring a recompilation of the program. (The User Interface shows only 5 layers by default but this can be changed with its calling parameters. See below.)

Layer description

Then, there is a series of lines for each layer (or material/color in the case of bitmaps). For each layer, the first line (in the example "`400 0 0.04977 1 1000000`" for the first layer) specifies the layer thickness t (Å), the layer thickness distribution Δt (Å), the atom density N (at./Å^3), the number of elements in the layer, and the number of projectiles to simulate this layer. If that last number is different form 0, it overrides the value specified in the beam description (second line of parameters in `corteo.in`). Parameters N and t are used to generate the appropriate factor of Eq. 32. Regarding Δt, the parameter is used to change the value of t before the simulation of each incident atom, according to a Gaussian distribution. This can be used to simulate the roughness of a layer, but it has to be understood that it does not account for lateral roughness, only its effect of layer thickness and corresponding widening of the energy spectrum, if that distribution can be assumed to be Gaussian. For simulations of actual roughness, one should rather consider bitmaps derived from microscopy.

Layer composition and collision partners

There are then, for each layer, a number of lines corresponding to the number elements in the layer. In the example, there is only one element in the layer, so there is only one line:

"`14 0 1 20 2 1 3 0.922297 27.97692653 0 0.046832 28.9764947 0 0.030872 29.97377017 0`".

The first two parameters are the element atomic number Z_i and mass M_i. If M_i is set to 0, the average atomic mass of the element is used. The next parameter is the fraction of this element in the layer. It does not need to be normalized to 1, normalization is carried out at program startup. This fraction is used both to select the collision partner during the ion slowdown process (Step 1.b.i) and is also used to compute the stopping power of the beam and recoil in the layer according to Bragg's rule and taking into account the compound correction. It is worth remembering here that the slowdown process is usually simulated as in this example, i.e. considering only one isotope of this element with an average isotopic mass. If one wants to simulate the ions slowdown process with the detailed isotopic composition, the same element would have to be included several times with each of its isotopes mass and fraction, as distinct elements of the layer.

The next two parameters ("`20 2`" in the example) are the displacement and surface energy used to compute full cascades. It is reminded that this type of simulation is still buggy and these parameters are not taken into account during IBA simulations.

The next parameter indicates if, as main collision partners, we will be simulating a natural abundances of the isotopes for this element. This is the case if this parameter is different from 0, and in the case where we are interpolating in tables, it will tell the program to issue a cross-section "stamp" indicating to use a cross-section table that consider itself a natural abundance of isotopes. Otherwise, the cross-section "stamp" will be such that a different cross-section table will be requested for each isotope. This parameter can be safely ignored (put any value) if cross-section tables are not used.

The next parameter indicates how many isotopes ("`3`" in the example) will be simulated as main collision partners. For each of these isotopes, 3 values have to be specified: the fraction of that isotope, its mass [amu], and whether or not (1 or 0) to use a cross-section table for that isotope.

Quick slowdown, multi-layers with same elements

When the target comprises only one layer consisting of one element, a special function is called to

compute the ions slowdown in which the steps of selecting the next collision partner and checking in which layer the ion currently is are skipped since they are unnecessary. This saves about 30% of the computation time.

On the other hand, it has to be understood that *if the same element with the same mass enters in the description of several layers, **it will not** increase the memory burden* in terms of scattering matrices since the same scattering matrix will be looked into when a collision occurs on the same element with same mass contained in different layers. Simulating multi-layers therefore does not necessarily require a much larger computational effort if the total number of distinct elements present in these layers is limited.

Absorber in front of the detector

After the complete target description, an absorber layer in front of the detector *must* be included. It is described exactly the same way as a target layer. If there is no absorber in front of the detector, the user has to include a dummy layer description of thickness 0 and containing at least one dummy element. That is what is done in the example with these two lines:

```
0 0 0 1 1 1
 1 1 1 1 1
```

3.1.4 Detectors

Then, the parameters associated with detectors are specified. Two detectors must be specified even if only one is used. RBS and ERD will use only the first detector. For RBS-TOF and ERD-TOF, the first detector is the first timing foil and the second one is the stop detector. In these cases, the second detector is specified in the reference frame of the first detector (see section 2.5.6). There are 11 parameters for each detector (in the example, "`0 -4.34e+08 1.90e+08 -4.33e+08 -4.34e+08 1.45e+08 -4.50e+08 -4.41e+08 1.42e+08 -4.43e+08 0`"). The first parameter is set to 1 to consider an elliptical detector (a circular detector being a special case of an elliptical one) and is set to 0 if the detector is rectangular. The next nine parameters specify the coordinates (in Å) of $\vec{p}_1$, $\vec{p}_2$, $\vec{p}_3$, the three points that define the detector rectangle. As explained in Section 2.5, $(\vec{p}_2-\vec{p}_1)$ must be perpendicular to $(\vec{p}_3-\vec{p}_2)$. The last parameter is the radius of the annulus (in Å), set to 0 if no annulus is considered. If the second detector is not used, just specify dummy values as in the example: "`0 -1 0 0 -1 1 0 -1 1 -1 0`".

The next line specifies the virtual detector size, that is, the factors by which the virtual detector is larger than the actual detector along its $(\vec{p}_3-\vec{p}_2)$ axis and along its $(\vec{p}_2-\vec{p}_1)$ axis, respectively.

3.1.5 Compound correction

The last series of parameters are related to the compound correction factors for the stopping power of the beam, and of each recoil in the case of ERD, according to Ref. 3. The number of traveling atoms was specified with the number of layers. For each if these ions, the Z is specified, and then the compound correction in each layer of material/color. If no correction is required, the factor is 1, and Bragg's rule is considered for computing the stopping power.

3.2 Bitmap

The target can be described by a 2D or 3D bitmap, but in order to avoid relying on external, system dependent libraries in Corteo (which is programmed in plain C), the bitmap that has to be provided in plain text mode. The format it the following: a series of lines of length corresponding to the dimension

of the bitmap in x as specified in `corteo.in`, and a number of lines equal to the dimension in y times the dimension in z. Vacuum is represented as a space character, and materials are represented by digits, starting with 0 for the first material, 1 for the second, etc. Here is an example of a 2D bitmap representing a nanoparticle of material 1 embedded in a matrix of material 0, with vacuum at the top, represented by white spaces on the left. The quote characters at the beginning and end of each line were added to indicate the beginning and end of each line.

```
“        00000000000000000000000000000000000000000000000000000000000”
“          00000000000000000000000000000000000000000000000000000000”
“            00000000000000000000000000000000000000000000000000000”
“            00000000000000000000000000000000000000000000000000000”
“            00000000000000000000000000000000000000000000000000000”
“            00000000000000111111100000000000000000000000000000000”
“            00000000000001111111111000000000000000000000000000000”
“            00000000000111111111111000000000000000000000000000000”
“            00000000001111111111111100000000000000000000000000000”
“            00000000011111111111111110000000000000000000000000000”
“        00000000000001111111111111111100000000000000000000000000"
“      000000000000001111111111111111111000000000000000000000000”
“     0000000000000001111111111111111111100000000000000000000000”
“     0000000000000001111111111111111111100000000000000000000000”
“    00000000000000001111111111111111111100000000000000000000000”
“    00000000000000001111111111111111111100000000000000000000000”
“    00000000000000001111111111111111111100000000000000000000000”
“    00000000000000001111111111111111111100000000000000000000000”
“    00000000000000001111111111111111111100000000000000000000000”
“    00000000000000001111111111111111111100000000000000000000000”
“    00000000000000001111111111111111111100000000000000000000000”
“     0000000000000000011111111111111111100000000000000000000000”
“     0000000000000000011111111111111111100000000000000000000000”
“      000000000000000000111111111100000000000000000000000000000”
“        00000000000000000000000000000000000000000000000000000000”
“        00000000000000000000000000000000000000000000000000000000”
“       000000000000000000000000000000000000000000000000000000000”
```

The material 0 is described by the first layer description in `corteo.in`, material 1 by the info regarding the second layer, and so on.

3.3 Output text and files

3.3.1 Screen output

If the execution ended without major error, here is an example of the screen output of the program.

```
8 thread(s).
Random seeds = 8550983 2017772465
Simulating RBS spectrum of 1.50 MeV 4He
beam incident at 45.0 deg. and detector at 46.9 deg. from surface normal.
beam and detector 15.0 deg. apart.

Layer 0: 20000.0+/-  10.0 A of  28Si (100.0%), N=4.977e22/cm3, Nt=9954e15/cm2
Absorber: not present.
  3 simulations
Running simulation 1 of 3: 28Si in layer 0
_____________________________________________________________________________
Done in 441 s, 856.8 coll/ion, 47.8 us/ion, 55.8 ns/coll
      9222970 total incident ions
         5366 detected atoms
      3456640 particles emerged at surface but were undetected
            0 main collisions below cutoff angle
            0 particles entered detection system but were finally not detected
          498 incomming particles abnormally exited at top
       837573 particles exited at target bottom
      4909210 particles implanted or below channel 0
```

```
0 particles generated in a direction not allowed by kinetics
0 particles not detected after correction for virtual detector
0 main collisions with vacuum in pixel target

0 pixels in a 3rd material between 2 collisions
5 reduced impact parameters below minimum value
```

The first line indicates the number of threads running. The subsequent lines describe in natural language some of the beam and target parameters. It namely helps to confirm that the angles are set properly. Computing all the matrices and tables takes typically a couple of seconds, and then, the simulation of each isotope in each layer starts, displaying a message that starts with "`Running simulation`". Then, each thread will issue a "_" after each 1/10th of the work done. If there are 4 threads, we expect 40 "_". The user interface simply counts the number of "_" to show the progress. The time needed for the spectrum simulation (excluding the initial matrices and tables computation) is then displayed with some grossly computed statistics about the average time needed to simulate the trajectory (in and out) of each ion and each collision. Then, some statistics about what happened to the simulated ions appear. The number of ions corresponding to "`particles implanted or below channel 0`" correspond to those ions that reached the minimum energy.

Finally, there could be a series of error message. (Note that normal output is sent to `stdout` while errors are sent to `stderr`. A scripting language controlling Corteo could therefore access them separately.) In this example, the last line specifies that 5 of the 856 millions of collisions computed had a reduced impact parameters that was below the minimum value permissible. No check of this value would have returned a bad index value to access the scattering table. Checking these values, though, slows the program by 15-20%.

3.3.2 File `corteo.detect`

In the case of IBA simulations, including Full RBS, Corteo writes a file named `corteo.detect` that is the equivalent of a list-mode file. Corteo first copies the content of `corteo.in` as a header of that file, with each line preceded by a "%" sign so it can be skipped as a comment line when reading the file with Matlab or Python. It then writes a comment line containing the header of each column, identifying the variables. Then, depending if the parameter "write details" is set to 0 or 1, Corteo will write a small or large number of values regarding each detection event. More details can be useful to understand what cause features in a spectrum, but it also produce large files. Beware!

IBA techniques except full RBS

The header should make the file self-explanatory, but to summarize, in the case of IBA techniques with "technique" parameter greater than 0 (i.e. excluding full RBS, see section 3.1.1), when parameter "write details" is set to 0, Corteo writes the energy at the detector, the cross-section (which can be a screened cross-section depending on the user settings), the parameter by which the cross-section should be multiplied to get a number of counts (i.e. the factor in Eq. 32), the energy of the incident ion just before the main collision and the cosine of the collision angle (these two parameters being useful to compute or interpolate our own cross-section, for instance), the cosine of the angle the particle leaving the main collision (scattered or recoil) makes with the detector (useful to estimate the with of the cone angle), the depth of the main collision, the Z of the collision partner, the layer or material in which the main collision occurred, and in the case the user is to use tables to interpolate the cross-section, a stamp consisting of a string indicating the isotope mass and Z of the collision partners.

In the case of coincidence, the line starts with the detected energy of the other particle detected. (An

event is written in the file only if both particles intersect the detector.)

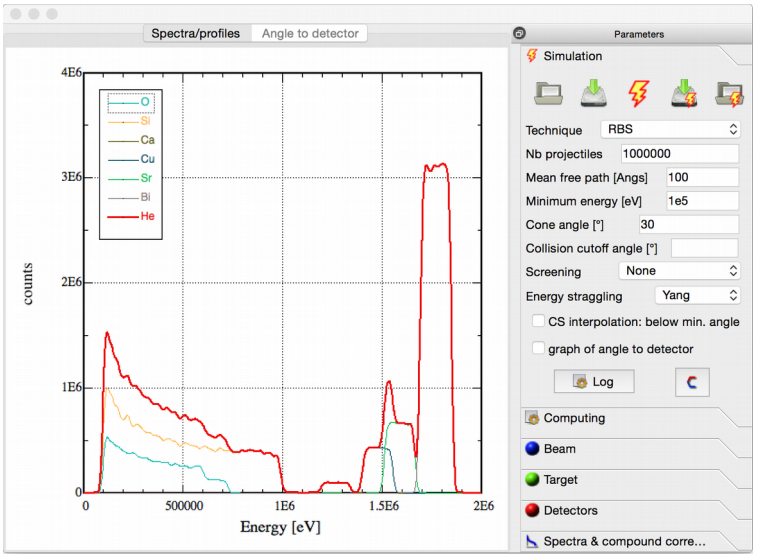
If parameter “write details” is set to 1, Corteo adds to the line other values such as the minimal approach distance during the main collision, the locus of the main collision and the unit vector $\hat{u}$ of the incident ion before the collision and that of the particle(s) leaving the collision locus, the location where the detected particle left the target and the unit vector $\hat{u}$ at that location, and the coordinates of the impact point on the detector and unit vector $\hat{u}'$ in the detector reference frame. If and absorber is specified, the location and vector $\hat{u}'$ as the particles leave the absorber is also written. In the case of coincidence, all the info regarding the second particle leaving the collision site is also written.

Full RBS

Full RBS is a special case where the simulation is actually that of an implantation or irradiation where Corteo checks if the ion intersects the detector. The main collision approximation is not involved (actually, the main collision happens randomly somewhere along the trajectory), and a spectrum is obtained directly by counting +1 for each particle intersection the detector.

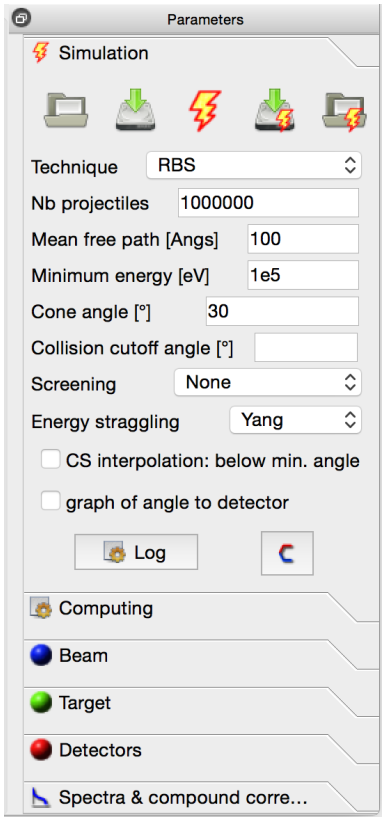
So in the case of Full RBS, if “write details” is set to 0, the `corteo.detect` contains only the energy and the cosine of the maximum scattering angle. If “write details” is set to 1, Corteo additionally writes, for each event, the location where the detected particle left the target and the unit vector $\hat{u}$ at that location, the coordinates of the impact point on the detector and unit vector $\hat{u}'$ in the detector reference frame, and the cosine of the five largest scattering events with their corresponding depth. (The number five is hard-coded but can be changed easily. Such a change requires recompiling the code.)

4 Corteo User Interface



CorteoUi is the Graphical User Interface of Corteo. As mentioned in the previous chapter, Corteo is only an executable program that carries out the simulations, namely on a computer cluster. An input file needs to be prepared. Most parameters could be changed by hand (a difficult one being the coordinates in space of the detector(s)) and the results could be read with a scripting language such as Python or Matlab, but it is indeed much more convenient to carry this out with CorteoUi. Given a beam, detector(s) and target description, CorteoUi calls Corteo to run the simulation, and produces a spectrum from the Corteo output. The program allows the user to set most parameter and access most features of Corteo. This includes the simulation of isotopes, plotting the angular spread of outgoing ions to determine the optimal cone angle, etc. (Isotopes can be simulated as main collision partner while not entering target description for slowdown computation; see Sections 2.3 and 2.4 for a detailed explanation of the impact of including several different masses in Corteo's target description.)

Fig. 14 shows an example of the interface. The main window consists of a graph in which are plotted the spectra resulting from the simulations and where experimental spectra can be loaded for comparison. (In that particular case, CorteoUi also generated a sub-spectrum for each element.) On the right are a series of panels where the user sets all the simulation parameters. In the following sections, we will describe each of these panels, and then how operations and modifications can be made in the graphs.



4.1 Simulation panel

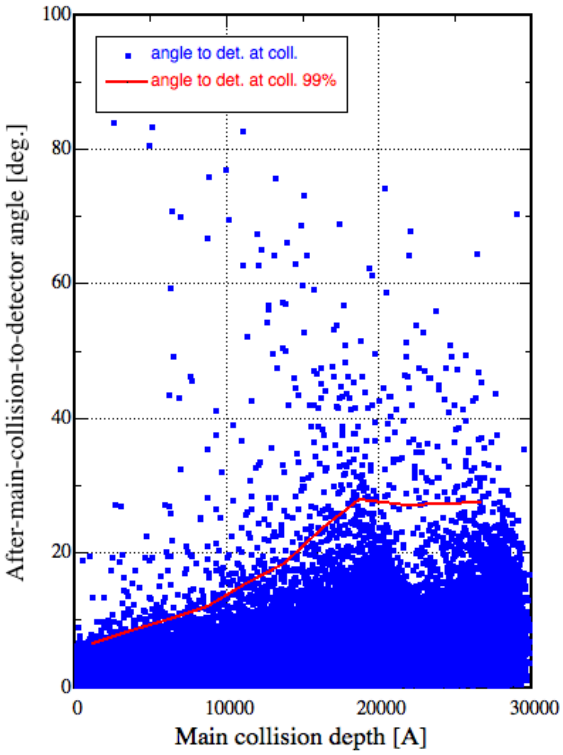
The first panel that shows up at program startup (Fig. 15) is a panel that contains simulation-specific parameters, i.e. the parameters associated with the simulation itself rather than the beam, target or detectors description. The five buttons at the top are used to load or save files. The first to buttons identified are used to load or save the simulation parameters (i.e. all panels content) from or to a file in XML format, with extention `.cml`. They do not save the content of the graph on the left. This must be done separately as explained in Section 4.7. Once all parameters are set properly, the user clicks the third button (lightning symbol) to run a simulation. The forth button saves the corteo.in file without running a simulation, so the user can transfer the file to a computer cluster to run the simulation. The last of these 5 buttons is used to read the simulation output. It has at least two uses: the

user can click it to read the results of a simulation ran on a computer cluster, and it can be used to re-read the output of the last simulation. For example, if the user wants to change the detector calibration or resolution, there is no need to run the simulation again since the simulation output is in the form of an event list, and the spectrum is accumulated from this list based on the current detector settings.

The technique is selected from the selection box "Technique". This will affect the content of the panels, some options being enabled or disabled depending on the selected technique. The other parameters are well identified and the user can over the fields with the mouse to see a tooltip appear. The user can also read the previous chapters to get their exact meaning.

Note specifically that the *number of projectiles is the number of ion that will be simulated for each element in each layer, and has noting to do with the incident charge* (except in the case of Full RBS). If this number is set to 10^6 and there are three layers 3×10^6 ion trajectories will be simulated.

The "Log" button pops up a window containing Corteo's screen output (see Section 3.3.1) of all sub-simulations ran by CorteoUi since its installation. It namely provides info about errors. A new log can be created by deleting or renaming the file `CorteoUi.log` in the CorteoUi directory.

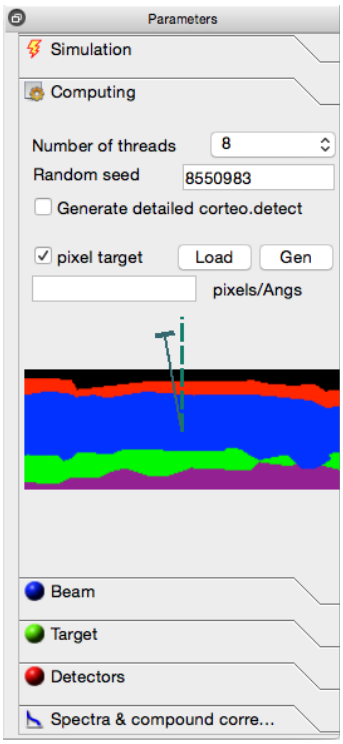


4.1.1 Finding the right cone angle

Just above the "Log" button, a check box allows the user to request a plot of the angle that each detected particle had with respect to the detector axis when they left the main collision. Because of MS, particles initially heading away from the detector may reach it. This graph helps to find how wide (what angle) should be the cone in order to have enough of these trajectories to reproduce MS effects while note making the cone uselessly wide and loosing a lot of particles away from the detector.

The graph is accessible by clicking the tab "angle to detector" at the top of the graph area. An example is shown in Fig. 16 for 2 MeV He backscattered from SiO_2, considering a cone angle of 90° (i.e. ions scattered in a half-sphere about the detector during the main collision). Each dot indicates the angle a detected particle made with respect to the detector when it backscattered during the main collision, as a function of the depth of that collision. There were 26 000 detected particles. The solid line includes 99% of the events with the smallest angle. This means that 99% of the scattered ions that were detected left the main collision within a cone of about 28° about the detector. In the 1% remaining (260 detected atoms in this particular case), one can see that some event left the main collision with an angle up to almost 90° about the detector, but were deflected back towards the detector by MS. Hence, by selecting a cone angle of 30°, one would consider 99% of the events, but would filter out those collisions with a smaller scattering angle, hence the largest cross-section, hence the largest contribution in Eq. 31. User should be careful if simulation of plural scattering is an issue.

4.2 Computing panel



The computing panel (Fig. 17) contains extra parameters related to how the simulation is ran. One can namely set the number of threads (depending on the number of cores the user wants to use on a processor, and a random seed. Note that if the user wants to run the program on several processors, the number of threads should still be set to the number of core per processor (or two times if hyperthreading is available), and one job per processor is ran, each with a different random seed. (See the end of section 2.2 and the beginning of section 3 to know how to do it.) If the box "Generate detailed corteo.detect" is checked, it sets the parameter "write details" to 1 in the Corteo input file. (See section 3.3.2. Watch out, it may produce Gb of data.)

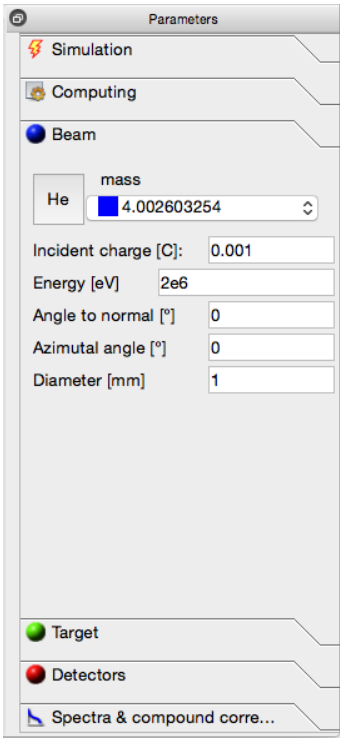
Then, since this panel was realtively empty, the author of Corteo found that it was a convenient place to allow the user to load bitmaps. To do it, the user checks the "pixel target" box, which allows the user to click on the "Load" button to load a `.bmp`, `.gif` or `.png` images for 2D description of the target, or `.xraw` files generated by MagicaVoxel for 3D description of the target. In the case of 2D images, this specifies the structure in the x-y plane (where x is the depth). The image can be derived from a micrograph as in the bitmap of Fig. 17. Since Corteo assumes periodic boundary conditions in y and z, a particle leaving the bitmap from the right is assumed to re-enter from the left. Since it is a 2D image, the target is assumed to be an infinite extrusion of the image in the z direction, perpendicularly to the image. In the case of 3D `.xraw` voxel maps, a scrollbar appears at the bottom in order to navigate along the $\hat{z}$ axis in the image, and periodic boundary conditions are assumed in both the z direction (perpendicular to the screen) as well as along the $\hat{y}$ (lateral) axis.

The loaded image should contain a limited number of color, since each color corresponds to a different material for which the elements and proportions have to be specified. Note that color **black** is reserved for **vacuum**, so a rough of nano-structured sample sample has to be drawn on a black background, as shown at the top of the bitmap in Fig. 17.

The purpose of the "Gen" button is to generate a bitmap from a file containing the x, y coordinates of a depth profile. This can be for example a diffusion implantation profile, or a complex stack of two materials. That file must be a 2-column file with an x and y value separated by a space on each line. The depth scale (x values) has to start at 0, be regularly spaced, and in increasing value order. Concentration (y values) has to be between 0 and 1. It will produce a 2D bitmap containing two colors. The number of vertical pixels will correspond to the number of lines in the file, and the user is asked for the image width. CorteoUI then generates an image with a random number of pixels of one color that correspond to the concentration at that depth.

CorteoUi superimposes the projection on the x-y plane of the beam axis (represented by a dashed line) and the detector axis (solid line). The points $\vec{p}_1$, $\vec{p}_2$, $\vec{p}_3$ defining the detector plane are also plotted with a line relating them. In Fig. 17, this appears as a dash perpendicular to the detector axis, since the detector surface is perpendicular to the x-y plane.

4.3 Beam panel



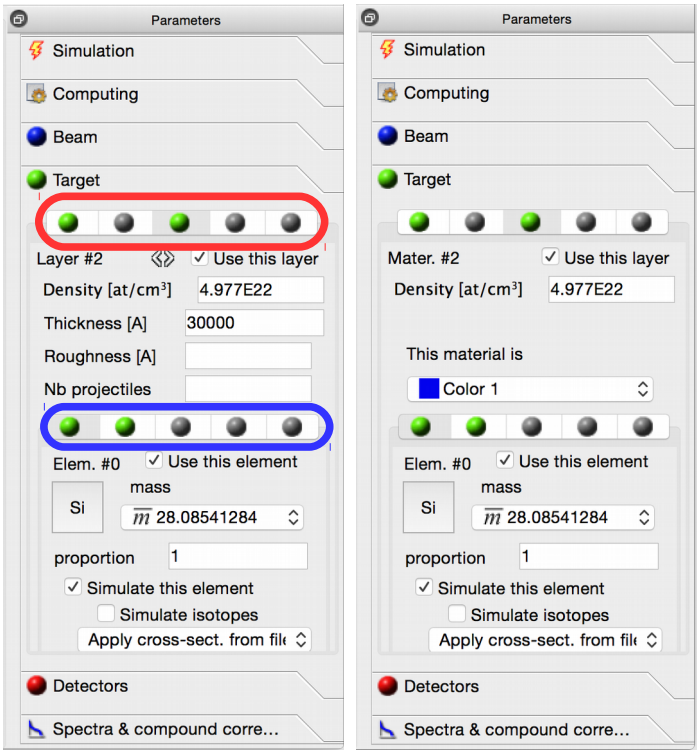
The beam panel is used to set the beam parameters. Clicking the button (here identified He) will open a periodic table from which to select the beam element. Selecting a beam will automatically update the mass selection box that contains only stable or long-lived isotope masses. Next to each mass, the blue square darkness is proportional to its relative abundance of that isotope; pale blue or almost white indicates a rare isotope. The selection box also contains an entry identified $\bar{m}$ which is the average atomic mass for this element.

The incident charge is actually the "atomic charge", i.e. the number of incident ions times e, without regard for the charge state of the ions. (It's q in Eq. 31, expressed in coulombs.) As a matter of fact, this parameter is not passed to Corteo during the simulation. It is only used afterwards when producing the spectrum from the list of detection events. Hence, the charge can be changed and the simulation results reloaded to generate a spectrum with the new charge, using the 5th button at the top of the "Simulation" panel, and without having to run the simulation again.

When an "angle to the normal" greater than 0 is set, the beam is tilted towards the $\hat{y}$ axis (see Fig. 20). The azimuthal angle allows a rotation of the beam around the $\hat{x}$ axis.

The beam diameter is actually two times the standard deviation of a Gaussian, radially symmetric beam.

4.4 Target panel



The target description panel is slightly different depending if the target is a layer stack (usual case, Fig. 19, left) or if it is represented by a bitmap (pixel target checked and bitmap loaded in the Computing panel, right panel in Fig. 19). The different layers or materials (corresponding to each color of a bitmap) are accessed by the tabs circled in red in Fig. 19.

In the case of a layered target, the top layer is be the leftmost tab, #0. A layer can be used or unused depending if the check box under the tabs is checked. Used layers feature a green "light" in their tab while unused layer feature a gray "light". Unused layers have their content disabled. In the left example of Fig. 19, two layers are used so the target consists of two layers. The purpose of this used/unused mechanism is to be able to define layers, for example, surface contamination, and pounder their effect on the simulation when they are present or not, while not erasing the description of these layers. The layer order can

be changed using the arrows (< >). Entry fields allow setting the layer thickness (in Å) and density (in at/cm^3), as well as the roughness (that is, Gaussian a thickness distribution). A number of projectiles was specified in the first panel, but another number can be specified for that layer, which overrides the value set in the first panel.

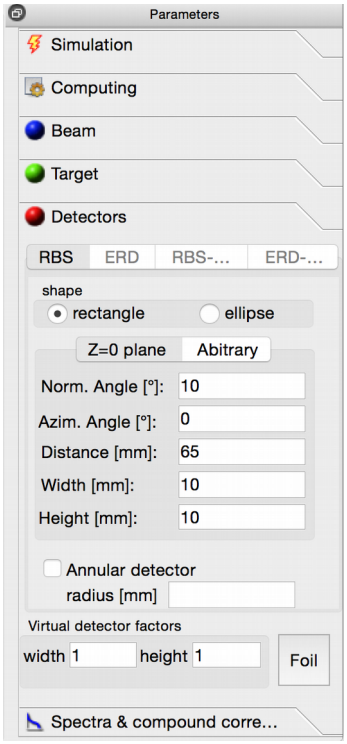
In the case of a target described by a 2D or 3D bitmap, the user has to set the density and select a color. Each color of the bitmap (except black which is reserved for vacuum) must be assigned to a material. In addition, the user can specify that a material is related to a substrate layer, and in this case, must specify its thickness.

Then the elements contained in each layer or material/color are set in the series of tab circled in blue in Fig. 19. As for the layers, an element can be used or unused, its tab featuring a green or gray "light" accordingly. The element atomic number and mass are set the same way as for the beam: clicking on the button (here identified Si for silicon) pops up a periodic table from where the element is selected. Selecting an element updates the isotope mass selection box, including an entry identified $\bar{m}$ which is the natural average atomic mass for this element. The proportion of this element in the layer is set in the entry field below the mass selection box. The proportions need not to sum to 1, normalization is carried out automatically by Corteo.

In the case of an IBA simulation, two check boxes appear. The first one is checked if the user wants that a spectrum is simulated for this element. For instance, substrate simulations can be relatively long and can be avoided during initial optimization of surface peaks, then included in the final simulation. If the element is selected for simulation, the user may want to simulate each of its stable/long-lived isotopes. By checking "Simulate isotopes", the number of simulated projectiles will be divided in proportion of the natural abundance of each isotope.

4.5 Detectors panel

In this panel, the detectors shape, position and size are set. In the case of RBS, only the first tab is enabled, while only the second tab is enabled for ERD. For TOF techniques, a second tab is enabled in order to define the stop detector. (The start detector is still defined by the first tab.) In the case of coincidence, both the RBS and ERD tabs are enabled since both the scattered and recoiled ions are detected.

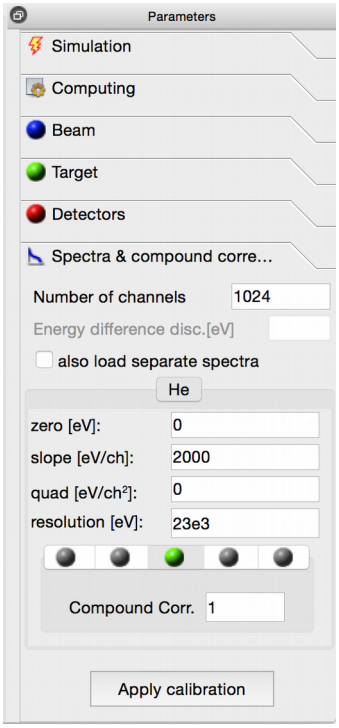


The shape of each detector is either rectangular or elliptical. Then, the user would normally have to set the coordinates of $\vec{p}_1$, $\vec{p}_2$, $\vec{p}_3$ defined in Section 2.5. This is required if the detector normal does not point towards the impact point of the beam and is possible to set directly the coordinates by clicking on the tab named "Arbitrary". Usually, the detector surface normal points toward the beam spot and the detector can be defined more simply in the "Z=0" tab by setting its angle from surface normal (towards the $\hat{y}$ as for the beam in the Z=0 plane, see Fig. 20). Additionally, the detector can rotated about the $\hat{x}$ axis (and is actually in the Z=0 plane only if the azimuthal angle is set to 0 or 180º). The user also sets the detector distance from target origin, its width ($\hat{y}$ direction) and it height ($\hat{z}$ direction) in millimeters. Circular and square detectors have the same height and width. The detector can feature an annulus (check box) for which the radius is specified.

Separately from detectors definition, the virtual detector enlargement factors are set in the identified group box. These factor apply to the first

detector in the case of TOF techniques, and is not applicable to coincidence (see Section 2.1). A foil can be defined in front of this first detector (either an absorber or a timing foil) by clicking the "Foil" button. A window containing a form similar to those for layer definition will pop up. The check box "use this layer" must be checked in that window in order for the foil to be considered in the simulation.

4.6 Spectra panel



The last panel (Fig. 21) is used to set all things related to calibration and spectrum generation. The number of channel must be large enough to contain the maximum energy, according to the calibration. It is worth noting that contrarily to most parameters in the previous panels, most parameters in this panel do not affect the simulation. It only determines how the information provided as a list by Corteo will be classified into a spectrum or spectra. It also carries out the convolution by the energy resolution. Hence, the user can change these parameters and reload the results from the last simulation by clicking the fifth button at the top of the Simulation panel. Still, two parameters in this panel do influence the simulation: i) the energy difference discrimination in the case of coincidence, and ii) the compound correction parameter[16] to the stopping power (one for each element traveling through each layer).

The rule is that a different spectrum will be loaded for each *detected element*. In the case of RBS, only the beam is detected and CorteoUi shows one spectrum, although it can also load one spectrum for each scattering element by checking the "also load separate spectra" box. In the case of ERD, there will be one spectrum per detected element and a calibration can be specified for each detected element by selecting the corresponding tab. This included the detector energy resolution. In the case of TOF techniques, a flight time spectrum will be generated (according to the flight length and speed between the start and stop detectors) and a time calibration has to be entered with a time resolution.

Then, a compound correction for the beam and each recoil can be specified for each layer. Although this seems a strange place for this parameter, this series of tabs actually contain the list of each ion for which the transport in the material will be simulated, and for this purpose, a compound correction may be considered. For this reason, a tab corresponding to the beam is always included so its compound correction can be specified as well.

Finally, a button named "Apply calibration" appears at the bottom of this panel. By clicking on it, the calibration shown is applied to the X axis of the currently selected dataset in the main graph. This is useful if the user has loaded an experimental spectrum for which the X scale is in channels and wants to apply the indicated calibration. It is apply as $x = \text{zero} + i \cdot \text{slope} + i^2 \cdot \text{quad}$ where i is the number of the point in the list, starting at i=0 for the first point. The y value remains unchanged.

4.7 Graphs

Several operations and modifications can be made in the graphs and the data they contain. Most of these modifications are accessible by right-clicking on one of the components of the graph.

- From within the graph area but excluding the legend area:
  - *Load*: data from an ASCII file can be loaded into the graph (e.g. an experimental spectrum) by right-clicking inside the graph area elsewhere than in the legend and selecting “Load” from the menu. At the bottom of the dialog that shows up, the number of lines to skip (e.g. comment lines) at the beginning of the file can be specified, as well as the data columns to read as X and Y. If 0 is specified as a column number, this data will be replaced by the line number. Columns must be separated by spaces. Data not convertible into a number are replaced by 0.
  - *Zoom*: one can zoom a graph by left-clicking and dragging the mouse within the graph area. (Actually, the zoom has to start within the graph zone but can extend outside as to extend the viewing area.) Unzooming is obtained by right-clicking inside the graph area elsewhere than in the legend, and selecting “Unzoom” from the menu.
  - *Markers*: markers can be defined (as gray zones in the graph area) by left-clicking and dragging the mouse within the graph area while holding the Control key. If such markers are defined, operations described below will applied only to the data of a dataset within the markers. If no markers are defined, operations are applied to all the data of a dataset.
  - *Legend style*: A dialog to set the position, line style and font of the legend can be accessed by right-clicking inside the graph area elsewhere than in the legend and selecting “Edit legend” from the menu.
  - *Copy to clipboard*: The graph can be copied as a bitmap into the clipboard by right-clicking inside the graph area and selecting “Image to clipboard” from the menu.
- From the axis area:
  - *Log/lin*: by right-clicking inside an axis labeling area (outside the graph area), one can toggle between a logarithmic and linear axis by selecting the appropriate choice from the menu.
  - *Axis style and min/max*: by right-clicking inside an axis area and selecting “Edit axis” from the menu, a dialog containing different tabs opens where the axis title, its font, its color and its position can be set, as well as the font, color and position of axis labels, the color and style of the frame, and the number of ticks and grids as well as their color and line style. The user can set the minimum and maximum of the axis in the label tab of this dialog. The minimum and maximum will actually be rounded off according to the number of major ticks, and the number of ticks adjusted to have “nice” numbers.
- From the legend area, one can carry out several operations on datasets themselves. By right-clicking on the name of a dataset in the legend and selecting:
  - *Save*: one can save the current dataset in an ASCII file. A file save dialog asks for a filename. The dataset legend entry will be changed to the name of the saved file.
  - *Copy*: a copy of the current dataset is added to the graph. Its name will be extended with an incrementing number.
  - *Delete*: the current dataset is deleted without warning.
  - *Edit*: the user can change the dataset name, its line style and color, and its symbol face, size and color.

- *Sort*: the data are sorted in increasing order of their X values.
- *Swap*: the X and Y data are swapped.
- *Operation*: the user can add, subtract, multiply or divide the dataset by a constant (which affect its X and/or Y data as selected) or by another dataset (affects only Y data).
  - If markers are defined, only the data included in the markers will be affected.
  - Operations can be useful to scale data by a factor, or multiply them by a correction curve such as a detection efficiency or a charge fraction.
  - When carrying out an operation with a dataset, if there is no one-to-one correspondence between the datasets, the data are interpolated linearly. An error message will show up if the range of the selected dataset does not cover the the range of the marked data.
- *Fit*: the data are fitted or smoothed by the desired function. The order represents the order of the polynomial fit, the width of the boxcar average in number of points, or the number of splines (>4).
  - If the data are plotted on a logarithmic axis, the fit is applied to the logarithmic value of the data. This is useful to carry out an exponential or power law fit to the data.
  - If markers are defined, the fit will be carried out only on the data included in the markers. The user can then decide if the fit function should be used for interpolation between the markers or to extrapolate on the full data range. This can be useful for example when the user wants to fit a background and extrapolate the fit to the whole dataset in order to carry out a subtraction of this background.
- *To clipboard*: the dataset is copied to the clipboard as two columns separated by a tabulation. Such data can be pasted into a text file or in a spreadsheet.

There is unfortunately no “undo” function implemented at the moment, and no warning appears if the user is going to loose data by deleting them or closing the application. If a significant amount of work has been necessary to obtain some data, the user is urged to save the spectrum before doing anything else. Note that the output of the last simulation can be read without having to run the simulation again. Just hit the fifth button at the top of the Simulation panel. This can be practical if the user wants to tweak the calibration parameters or see the effect of adjusting the energy resolution of the detector.

5 Conclusion

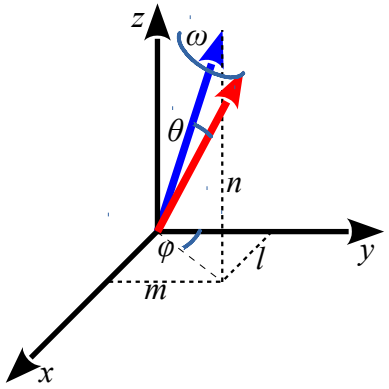
Monte Carlo simulations are a useful tool to add to the IBA practitioner arsenal, in complementarity to analytical simulation programs, namely to better account for the effects of MS. This document presented the theory behind MC simulations of IBA spectra with Corteo. In addition to the RPA, CPA and BCA approximations, Corteo uses other approximations including an adjustable mean free path (to use with care), the “main collision” approximation, and scattering angle components and stopping powers extracted from tables without interpolation thanks to the binary representation of floating point numbers. This makes possible the simulations of an IBA spectrum including the trajectory of a million ion within a few seconds.

Care was taken to keep the simulation as general as possible. This allows the implementation of several techniques, provided that the cross section of the main scattering events can be computed at all angles or extracted from a table. It is also possible to define arbitrarily oriented and positioned detectors. Several data can be generated for extra data treatment. The Corteo simulation engine itself concentrates on simulations and does not feature a user interface, making it runnable on any platform featuring a C compiler. Its output can be read by a scripting language such as Python or Matlab.

The user interface CorteoUi can be used to control most of Corteo's feature and properly accumulates the different parts of a spectrum. It can be used to generate the Corteo input file more easily, namely in regard of the detector location.

The program is distributed under the terms of the GPL[17] and can be downloaded free or charge and modified, provided that redistribution of the modified version of the program is made easily available to anyone, including the source code of the modified version.

APPENDIX I. Directional vector rotation



I.1 Rotation

Let $\hat{u}=l\,\hat{x}+m\,\hat{y}+n\,\hat{z}$ be a unit vector with l, m, n the directional cosines: $l^2+m^2+n^2=1$. We want to tilt it by an angle θ to obtain $\hat{u}'$, and then rotate it by an angle ω around the axis defined by $\hat{u}$ to finally obtain $\hat{u}''$.

For the θ rotation, we apply rotation matrices for Euler angles ϕ and θ in the x-convention, following Goldstein[17]. For the ω rotation, we then use the *rotation formula*[18].

<u>θ rotation</u>

1) Frame rotation (in clockwise direction) by an angle ϕ about axis $\hat{z}$ so the $\hat{y}$ axis is parallel to projection of vector $\hat{u}$ on x-y plane. This is carried out by the application to vector $\hat{u}$ of rotation matrix D for an angle $\varphi=-\arctan(l/m)$

$$D=\begin{pmatrix}\cos\varphi & \sin\varphi & 0\\ -\sin\varphi & \cos\varphi & 0\\ 0 & 0 & 1\end{pmatrix} \qquad \text{Eq. I.1}$$

We can also express $\cos\varphi=\dfrac{m}{\sqrt{l^2+m^2}}$ and $\sin\varphi=\dfrac{-l}{\sqrt{l^2+m^2}}$, $\sqrt{l^2+m^2}=\sqrt{1-n^2}$.

2) Rotation of vector $\hat{u}$ about the axis $\hat{x}$ by the desired θ angle: application of matrix

$$C=\begin{pmatrix}0 & 0 & 1\\ 0 & \cos\theta & \sin\theta\\ 0 & -\sin\theta & \cos\theta\end{pmatrix} \qquad \text{Eq. I.2}$$

3) De-rotation of frame by angle ϕ about axis $\hat{z}$ in counter-clockwise direction: application of inverse matrix

$$D^{-1}=\begin{pmatrix}\cos\varphi & -\sin\varphi & 0\\ \sin\varphi & \cos\varphi & 0\\ 0 & 0 & 1\end{pmatrix} \qquad \text{Eq. I.3}$$

Thus, $\hat{u}'=D^{-1}CD\,\hat{u}$ with

$$D-1\,CD=\begin{pmatrix}\cos^2\varphi+\sin^2\varphi\cos\theta & \sin\varphi\cos\varphi(1-\cos\theta) & -\sin\varphi\sin\theta\\ \sin\varphi\cos\varphi(1-\cos\theta) & \cos^2\varphi+\sin^2\varphi\cos\theta & \cos\varphi\sin\theta\\ -\sin\varphi\sin\theta & -\cos\varphi\sin\theta & \cos\theta\end{pmatrix} \qquad \text{Eq. I.4}$$

A few terms canceling out, we find $\hat{u}'=l'\,\hat{x}+m'\,\hat{y}+n'\,\hat{z}$ with

$$l'=l\left(\cos\theta+\frac{n\sin\theta}{\sqrt{1-n^2}}\right),$$

$$m'=m\left(\cos\theta+\frac{n\sin\theta}{\sqrt{1-n^2}}\right), \text{ and} \qquad \text{Eqs. I.5}$$

$$n'=n\cos\theta+\sqrt{1-n^2}\sin\theta$$

ω rotation

Using the *rotation formula*,[17] we determine

$$\hat{u}" = \hat{u}' \cos\omega + \hat{u}(\hat{u}\cdot\hat{u}')(1-\cos\omega) + (\hat{u}'\times\hat{u})\sin\omega \qquad \text{Eq. I.6}$$

which is the rotation of vector $\hat{u}'$ by an angle ω about the axis defined by vector $\hat{u}$. Knowing that $\hat{u}\cdot\hat{u}' = \cos\theta$, $\hat{u}" = l"\hat{x} + m"\hat{y} + n"\hat{z}$ has the following components:

$$\begin{aligned} l" &= l'\cos\omega + l\cos\theta(1-\cos\omega) + (m'n - n'm)\sin\omega, \\ m" &= m'\cos\omega + m\cos\theta(1-\cos\omega) + (n'l - l'n)\sin\omega, \\ n" &= n'\cos\omega + n\cos\theta(1-\cos\omega) + (l'm - m'l)\sin\omega \end{aligned} \qquad \text{Eqs. I.7}$$

Replacing prime coefficients (Eqs. I.5), another few terms cancel out and we find

$$\begin{aligned} l" &= l\left(\cos\theta + \frac{n}{\sqrt{1-n^2}}\sin\theta\cos\omega\right) + m\sin\theta\sin\omega\underbrace{\left(\frac{n^2}{\sqrt{1-n^2}} + \sqrt{1-n^2}\right)}_{1/\sqrt{1-n^2}} \\ m" &= m\left(\cos\theta + \frac{n}{\sqrt{1-n^2}}\sin\theta\cos\omega\right) - l\sin\theta\sin\omega\underbrace{\left(\frac{n^2}{\sqrt{1-n^2}} + \sqrt{1-n^2}\right)}_{1/\sqrt{1-n^2}} \\ n" &= n\cos\theta - \sqrt{1-n^2}\sin\theta\cos\omega \end{aligned} \qquad \text{Eqs. I.8}$$

Defining $k = \sqrt{1-n^2}$ and regrouping terms we finally find, for a tilt of $\hat{u}$ by angle θ followed by a rotation of the resulting vector by an angle ω about $\hat{u}$, the following directional cosines

$$\begin{aligned} l" &= l\cos\theta + \frac{\sin\theta}{k}(l\,n\cos\omega + m\sin\omega) \\ m" &= m\cos\theta + \frac{\sin\theta}{k}(m\,n\cos\omega - l\sin\omega) \\ n" &= n\cos\theta - k\sin\theta\cos\omega \end{aligned} \qquad \text{Eqs. I.9}$$

which are the same as Eqs. 20. These are the formulae used in TRIM's `DIRCOS` routine to compute ions trajectory deflection once $\sin^2\Theta_{CM}/2$ is determined using the `MAGIC` algorithm and converted to laboratory coordinates through Eq. 20. In Corteo, the value of $\cos\theta$, $\sin\theta$, $\cos\omega$ and $\sin\omega$ comes from matrices and lists stored in memory. $k = \sqrt{1-n^2}$ is computed with the single precision floating point C function `sqrtf`, which is found to provides excellent precision without waisting too much computing time.

Reference

1 Gnu General Public License (available at www.gnu.org/licenses/gpl.html)
2 See for example the Handbook of Modern Ion Beam Materials Analysis, edited by J. R. Tesmer and M. Nastasi Materials Research Society, Pittsburgh, 1995
3 J.F. Ziegler, J.P. Biersack, U. Littmark, The Stopping and Range of Ions in Solids, Pergamon, New York, 1985.
4 J. H. Barrett, Phys. Rev. 166, 219 - 221 (1968)
5 M.T. Robinson, I.M. Torrens. Phys. Rev. B9 (1974) 5008
6 J.P. Biersack and L.G. Haggmark, Nucl. Instr. Meth. **174** (1980) 257
7 N. Bohr, K. Dan, Vidensk. Selsk. Mat.-Fys. Medd. **18** (1948) 8
8 W.K. Chu, Phys. Rev. **A13** (1976) 2057
9 Q. Yang, D.J. O'Connor, Z. Wang, Nucl. Instr. and Meth. **B61** (1991) 149
10 P. Sigmund and K.B. Winterbon, Nucl. Instr. and Meth. **119** (1974) 541
11 E. Szilagyi, F. Paszti, G. Amsel, Nucl. Instr. and Meth. **B100** (1995) 103
12 P.N. Johnston, R.D. Franich, I.F. Bubb, M. El Bouanani, D.D. Cohen, N. Dytlewski, R. Siegele, Nucl. Instr. and Meth. B 161–163 (2000) 314.
13 K. Arstila, T. Sajavaara, J. Keinonen, Nucl. Instr. Meth. **B 174** (2001) 163
14 B. Yuan, F.C. Yu, S.M. Tang, Nucl. Instr. Meth. **B 83** (1993) 413
15 F. Schiettekatte, Nucl. Instr. Meth. **B 332** (2014) 404
16 J.F. Ziegler, J.M. Manoyan, Nucl. Instr. Meth. **B 35** (1988) 215
17 Goldstein, Classical mechanics, 1980, p. 146.
18 *ibid*, p.165