An Introduction to the Simulation Kit

M.A. Karolewski

Adair studied Navajo silversmithing, and noted that learning was almost entirely by means of observation. 'Tom [the silversmith] said that the Navajo learn by watching and then doing, following exactly as possible what they have seen their teachers do.' Furthermore, when a novice makes an error, the teacher insists upon his finishing the piece, rather than beginning anew.[1]

Author e-mail: `karolewski@alum.mit.edu`

Version 2.0, released March 1998.

This article is distributed with the *Simulation Kit* (a software package for classical dynamics simulation of atomic collisions). News about updates to the *Simulation Kit* (if any) will be carried at one or both of the following URLs during 1998:

`http://www.geocities.com/CapeCanaveral/Lab/7039/simkit.htm`
`http://www.ntu.edu.sg/home/akarol/simkit.htm`

Contents

1. Introduction

1.1. Objectives

The original purpose of this article was to cover the minimum of background theory and concepts needed by anyone who wishes to use classical dynamics simulation to model the behaviour of atomic collision systems using the *Simulation Kit* or a similar package. However, in the writing I found that all but the introductory chapters became very specific to the *Simulation Kit*, so I have changed the title to reflect this. This is not necessarily a bad thing, since the reader can find excellent discussions of simulation theory in the review literature (see below), and also nowadays on the World Wide Web.

The range of particle energies considered here is 1 eV to - 100 keV. The is the range of energies that can be simulated to varying degrees of realism by the *Simulation Kit*. The lower energy limit is imposed by many-body and quantum effects, while the upper limit is determined by both the treatment used for modelling inelastic effects, as well as the practical difficulty of containing fast projectiles in small targets. This energy range typically covers secondary ion mass spectrometry (SIMS) and ion scattering spectroscopy (ISS) as well as a number of less familiar techniques and processes. The discussion begins with a review of the theory of the binary collision, and very quickly moves on to the task of computer implementation of many-particle simulations. The treatment given here is more superficial and less rigorous than what is found in the standard texts, since its objective is to get the reader going on the simulations as quickly as possible.

The article’s current purpose, then, is to serve as a manual of demonstration projects for the *Simulation Kit*, and as a repository for other information that seemed out of place in an on-line Help system. To a large extent, the discussion throughout the article is built around the features offered by the *Simulation Kit*. Ideally, you should browse through this article before, or while, you attempt a serious evaluation of the *Simulation Kit*. Some of the diagrams are in colour, and may be better viewed on-screen than in a monochrome printed version.

The demonstration projects (other than the tutorial, which is not discussed here) are designed to illustrate a range of *Simulation Kit* tasks. Some of them need to be more thorough in scope in order to qualify as serious but nevertheless, the projects address real research-level problems.

The user will probably find that while it is easy to generate simulation data, it is usually more difficult to extract meaningful output in the form of scattering profiles, spectra and so forth. Given a body of simulation data, it is necessary to specify one's informational requirements both in terms of the particle records which should be ignored (i.e. the *filtering* operation), as well as the parameter which should form the basis of spectral analysis or averaging. Normally these operations entail 2 distinct steps (experience with databases will prove invaluable here). For example, before plotting a spectrum of reflected projectile energies, you first need to ensure that you filter out records that refer to (a) target particles; (b) projectiles that lodged in the target. The exact filtering procedure required will depend on what output you chose to write at simulation design time (in the Run file made by `Spider`). The subject of data processing is discussed in greater depth in the manual for `Winnow` (`winnow.doc`) which you can find in the `docs` directory of the Simulation Kit installation.

This article covers version 2.2 of the *Simulation Kit*, which differs from previous versions in that it supports the optional inclusion of inelastic effects (Chapter 9) and uses a system of user-editable flags to fine-tune simulation options (Chapter 10). The article is written under severe time constraints, and I cannot pretend that even this second edition is anywhere near complete. I would appreciate it if readers could point out any errors of fact or omission.

The best way to start learning about the *Simulation Kit* is to run the program `Snook` using the tutorial and example projects provided. Most of the projects can be run immediately by loading the input files into `Snook`. (With the `iciss` project, you first need to view the `readme.txt` file found in the `examples\iciss` project directory.)

1.2. Sources

A number of references to the primary and review literature are supplied in context throughout the article. Since I anticipate that some readers of this article will be research students from a variety of backgrounds, I have also given references to textbooks I have used profitably myself over the years for some general topics that may not have taken root during undergraduate days. The references may be found at the end of the article.

There are several texts and review articles which cover the central subject matter of this article especially well, which should be mentioned at the outset. These are by: Smith et al.,[2] Eckstein,[3] Mashkova and Molchanov,[4] Harrison,[5] Smith and Harrison,[6] Robinson[7] and Niehus *et al.*[8] The reader should get hold of as many of these key references as possible.

In this connection, I wish to express my gratitude to those workers in the field who took the trouble to send me reprints of their work, or to answer my questions on uncertain points.

1.3. Notational Conventions

Vector quantities are written in boldface (**r**) whereas scalar quantities are represented by an italic font (*r*), or sometimes plain font (r). I use *b* for the impact parameter (designated as *p* or *s* by some writers).

1.4. *Simulation Kit* Components

There is frequent reference throughout this article to the following computer programs: *`Spider`*, *`Snook`* and *`Winnow`*. `Spider` is a utility whose function is to design and generate the input data files used for the simulations. The simulation engine itself resides in the `Snook` program. `Winnow`, finally, is the program which processes the output data created by `Snook`, and turns this data into scientifically useful information (sputter yields, energy spectra etc.). It may be helpful to new users of `Winnow` to regard the `Snook` output files as database files; the purpose of `Winnow`, then, is to process user-defined queries to this database, and to present the information in an acceptable form.

Users of the *Simulation Kit* will undoubtedly have to learn how to use both `Spider` and `Snook`. `Winnow` can be ignored (except as a file conversion utility) if the user prefers his own spreadsheet or statistical program. The programs should first be studied by running them on the demonstration projects.

1.5. Computation Time

Serious simulations of atomic collision phenomena make heavy demands on current-day personal computers. Although individual simulations run quite quickly, the time problem arises because of the need to gather adequate statistics. Sputtering simulations need several hundred runs, while ion scattering (ISS) simulations need many thousands. Frequently, one runs a variety of similar simulations while varying one system parameter (e.g. the angle of projectile incidence). The total simulation time required for a parameter-variation 'experiment' of this kind is often on the order of 1 day for publication quality data (i.e. good statistics). Even survey scans may consume a morning or afternoon (although here you do have the choice of concentrating only on the main features at the survey stage).

The simulation engine, `Snook`, can coexist happily with other Win 95 programs, nor does the program need to be monitored as it runs. The best time-management strategy, therefore, is to run long simulation projects as batch jobs (in the background, so to speak). `Snook`'s program interface can be 'minimised' while the computer is used for other work. You can read about batch processing in `Snook`'s online Help; `Spider`'s gadgets menu has a utility for generating 'batch definition file' templates.

The speed of execution of `Snook` can be optimised by a correct setting of various of the simulation options parameters. Again, the reader is referred to the online Help. In brief, to speed up `Snook` you should: (a) disable screen output options; (b) minimise `Snook`'s program window; (c) close the Graph window (which displays atomic trajectories).

The author doubts whether ion-surface simulations will ever execute quickly in a package of this nature. As the speed of CPUs increases, so too do one's ambitions grow. A researcher who uses the *Simulation Kit* for sputtering studies in the 1990s is unlikely to be satisfied with the 600-atom targets that formed the basis of many classical papers in this field in the 1980s.

2. Classical Scattering Theory

2.1. Classical Scattering Theory

The classical scattering theory described in undergraduate textbooks deals predominantly with the binary collision, involving a two-body, radial potential, $V(r_{12})$, in which two interacting particles first approach each other, and later recede. (Good graduate-level text-book treatments of scattering can be found in Goldstein,[9] or in Landau and Lifshitz.[10]) The strength of the interaction depends only on the separation (r_{12}) of the two particles. The scattering terminology applies to forces which have a finite range and tend to zero as the separation increases. However, the decline of the potential with separation need not be monotonic.

More complex interaction potentials can be envisaged. For example, the interaction might depend on the relative orientations of two interacting objects (e.g. a point charge interacting with a fixed dipole), or it might depend on the relative orientations of three point particles (a so-called ‘three-body’ potential): potentials involving three (or more) bodies are required to rationalise the structures of small clusters in terms of potential theory, for example. The force associated with the interaction of charged particles with magnetic fields depends upon the velocity of the particle, and not just its position in the field: this fact is exploited in mass spectrometry, for example.

For the most part, the collisions discussed in this chapter (and in this article) will be *elastic* collisions. An elastic collision is one in which the sum of the kinetic energies of the particles after the collision, $E_1 + E_2$, is the same as it was before the collision, E_0 (see Fig. 1). *Inelastic* collisions are those in which energy is transferred to some internal mode of excitation (e.g. electronic excitations) or dissipated (e.g. by friction). Chapter 9 will describe the simulation of inelastic processes in atomic collision systems.

The purpose of the following review is not to work through the equations of classical scattering theory (which can be found in any mechanics textbook), but to summarise the objectives and achievements of the classical theory which are useful in the context of atomic collisions.

2.2. Kinematics of Binary Collision

The classical scattering problem can be analysed at several different levels.

At the first level, the kinematic analysis involves the application of conservation laws (energy, momentum) to the problem, and establishes what relationships hold between respectively the collisional energy transfer, the scattering angles and the relative masses of the participating particles (see Fig. 1 for explanation of symbols).[*] These kinematic results are obtained from a consideration of the asymptotic particle trajectories, so they are valid for any type of central scattering potential.

Scattering under the influence of a central potential V(r) takes place in a plane. The geometry of the scattering process is depicted in Fig. 1, where the target is initially at rest. This coordinate system is known as the Laboratory coordinate system (*Lab system* or *Lab frame*), to distinguish it from the Centre-of-Mass reference frame (*COM system* or *COM frame*) introduced for analytical convenience in the following section.

---

[*] These relationships are well-known to players of Snooker (from which Snook takes its name) and croquet.

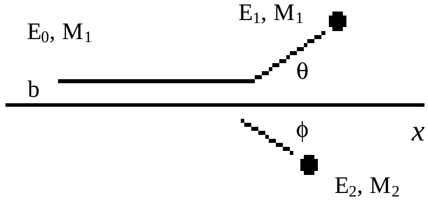

Figure 2.1. Scattering geometry for a binary collision. The projectile of mass M1, and energy E_0, approaches a stationary target of mass M2 located on the x-axis. After the interaction, the motion of the projectile is characterised by the scattering angle θ and its asymptotic energy E_1, while that of the target is characterised by the recoil angle ϕ and its asymptotic energy E_2 (= E_0 - E_1 in an elastic collision). *b* (the off-axis distance of the projectile trajectory) is the *impact parameter.*

The most useful equation produced by the kinematical theory is that which relates the collisional energy transfer to the projectile scattering angle in the Lab system:

$$E_1 = E_0(M_1M_2/(M_1+M_2))^2\{\cos\theta \pm [(M_2/M_1)^2 - \sin^2\theta]^{1/2}\}^2 . \qquad (2.1)$$

Equation 2.1 simplifies considerably if θ = 90° or 180°. This equation is the basis of ion scattering spectrometry (ISS), which involves the measurement of E_1 for fixed E_0, M_1 and θ, and thereby allows the mass M2 of the scatterer to be deduced.

For $(M_2/M_1) \geq 1$, only the positive sign before the $[(M_2/M_1)^2 - \sin^2\theta]^{1/2}$ term is applicable. Light projectiles are *backscattered* from heavy targets in small impact parameter collisions, whereas heavy projectiles push light targets ahead of them, and themselves only undergo small deflections.

The following remarks apply to a potential which falls off monotonically with separation. If $M_1 < M_2$, the projectile may be scattered over the entire range of θ (0-180°). However, for a heavy projectile ($M_1 \geq M_2$), the deflection angle θ increases as *b* increases, up to a maximum value given by $\sin(\theta) = M_2/M_1$. This trend is illustrated by the plot of trajectories shown in Fig. 2 (for argon scattering by a carbon atom target) which is taken from ref. [11].

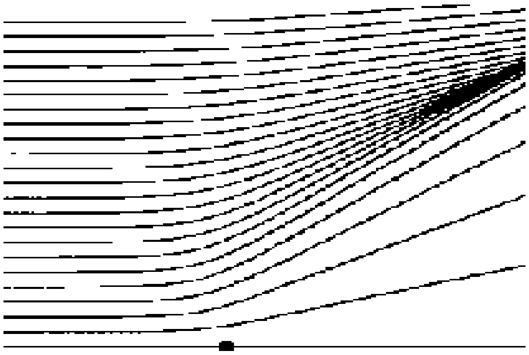

Figure 2.2. Trajectories followed by 1 keV Ar projectiles (mass 40) scattered from a carbon target (mass 12, initial location shown as black dot). The impact parameters of the various trajectories are separated by 0.05 Å.

The energy of the recoiling target particle is given by the kinematic theory as:

$$E_2 = E_0 * 4M_1M_2 \cos^2 \phi / (M_1+M_2)^2 . \qquad (2.2)$$

The angles θ and ϕ are related by a fairly complicated expression (it is easier to relate ϕ to the scattering angle in the COM system). According to equation (2.2), the maximum energy transferred to the target particle in an elastic collision, E_{max}, is:

$$E_{max} / E_0 = 4M_1M_2 / (M_1+M_2)^2, \qquad (2.3)$$

which occurs when the impact parameter is zero: a so-called *direct impact* (or *centre-to-centre*) collision. Equation 2.3 implies that energy transfers decline as the disparity in masses of the particles increases. For $M_1 = M_2$, $E_{max} / E_0 = 1$, whereas for $M_1 = 0.1M_2$ (or $M_2 = 0.1M_1$), $E_{max} / E_0 = 0.33$. In a proton-electron collision, $E_{max} / E_0 = 0.002$.

Gryzinski gives a very thorough review of the results of the kinematic theory, including the cases involving an inelastic energy loss or a non-stationary target. [12]

`Spider` uses the equations 2.1 and 2.2 to calculate the results shown in its ‘Scattering Relations’ window.

2. 3. Dynamics of Binary Collision

2.3.1. The Scattering Cross-Section

The dynamical theory of scattering is concerned with the determination of particle trajectories and scattering *cross-sections* for specific interaction potentials. To simplify the discussion we assume that $M_1 < M_2$, and that the potential falls off monotonically with separation.

Consider an experiment in which we measure the fraction of a projectile beam of incident intensity *I* which is scattered into an angular range θ to θ+δθ by two different target potentials $V_1(r)$ and $V_2(r)$. In the first case, this scattering will be associated with impact parameters between b_1 and $b_1+\delta b_1$, and in the second case with impact parameters between b_2 and $b_2+\delta b_2$. In other words, there is a distinct mapping $b \rightarrow \theta$ of a given impact parameter to a specific scattering angle for each type of potential. In three dimensions, the angular range θ to θ+δθ represents an annular-shaped solid angle δΩ, as depicted in Fig. 3.

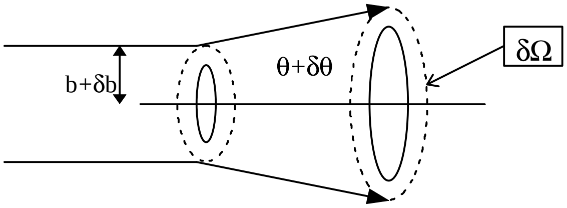

Figure 2.3. Depiction of a scattering process in three dimensions, showing how projectiles incident within an annulus defined by impact parameters in the range *b* to *b* +δ*b* are scattered into a solid angle δΩ.

If the incident particle flux impinging on the target has a uniform cross-section, then the number of incident particles in a small impact parameter range would be proportional to the area of an annulus of radius b and width $b+\delta b$, i.e $2\pi b\delta b$. Due to particle conservation, the number of particles $N(\Omega)$ scattered into the specified solid angle $\delta\Omega$ (representing the area of a detector) would also be proportional to the area of this contributing annulus of impact parameters. The scattering cross-section $\sigma(\Omega)$, which can be expressed as a function of θ (as well as E_0,M_1 and M_2), is the factor which relates the flux of incident particles to the number of scattered particles detected at a particular deflection:

$$N(\Omega) = I\sigma(\Omega)\delta\Omega \tag{2.4}$$

The concept of a scattering cross-section is useful when comparing the behaviour of a similar process in two different scattering systems: for example: 'the cross-section for 90° scattering is higher for Cu than for Al' (meaning, more scattered particles are detected for Cu at this angle).

If the potential is non-monotonic, or if $M_1 > M_2$, then scattering by an angle θ may be associated with more than one value of b (see Fig.2, for example). Effects of the first kind are important in determining the cross-sections of low-energy scattering processes, where they give rise to the *rainbow scattering* effect.[13]

The concept of a scattering cross-section is closely related to that of a *collision radius* (or alternatively, the *collision diameter*). Some physical processes (e.g. certain kinds of electron emission) can only be initiated if the collision impact parameter is below some critical threshold, b_{min}, known as the collision radius. The cross-section for such a process is equal to the area of the circle defined by a radius b_{min}. The cross-section concept is often replaced by the idea of a *yield* in solid-state scattering processes (e.g. the *sputter yield*, the *secondary electron yield* and so on): the yield is a measure of the average detection/emission probability for a signature particle associated with the process, referenced to the flux of stimulating particles (electrons per incident ion, etc.).

2.3.2. The Coulomb Potential

The Coulomb potential is described by an inverse square force law ($1/r^2$), which gives rise to a potential energy term of the form:

$$U(r) = Z_1 e\Phi = Z_1 Z_2 e^2/(4\pi\varepsilon_0 r), \tag{2.5}$$

where Φ is the electrostatic potential at a distance r from a point charge Z_2e, and Z_1e is the charge of the interacting projectile. (In a nuclear/atomic context, Z_1 and Z_2 represent atomic numbers, while e is the charge of the proton). ε_0 is the vacuum permittivity constant ($8.854\text{x}10^{-12}$ F m^{-1}). The functional form of Φ ($Z_2 e/(4\pi\varepsilon_0 r)$) is derived from elementary electrostatic theory.[14]

The equation of energy conservation for the Coulomb scattering problem in the Lab frame is:

$$\tfrac{1}{2} M_1\mathbf{v_1}^2 + \tfrac{1}{2} M_2\mathbf{v_2}^2 + Z_1 Z_2 e^2/(4\pi\varepsilon_0|\mathbf{r_1} - \mathbf{r_2}|) = E_0, \tag{2.6}$$

where $\mathbf{v_i}$ ($i = 1,2$) is the velocity of the ith particle ($= |\mathbf{i}v_x + \mathbf{j}v_y|$).

The goal of the analysis is to predict the scattering angle as a function of the impact parameter, *b*. The method of solution involves, as a first step, expressing the problem in a new system of coordinates ($\mathbf{R}$, $\mathbf{r_{12}}$) which are linear combinations of the coordinates ($\mathbf{r_1}$, $\mathbf{r_2}$) in the Lab frame; the first of these represents the position of the system centre-of-mass (COM):

$$\mathbf{R} = (M_1\mathbf{r_1} + M_2\mathbf{r_2})/(M_1+M_2), \tag{2.7}$$

while the relative particle position ($\mathbf{r_{12}}$) is given by:

$$\mathbf{r_{12}} = \mathbf{r_1} - \mathbf{r_2} \tag{2.8}$$

In terms of these variables, equation (2.6) can be expressed as:

$$½\ M\mathbf{V}^2 + ½\ \mu\mathbf{v_{12}}^2 + Z_1Z_2e^2/(4\pi\varepsilon_0 r_{12}) = E_0, \tag{2.9}$$

where $\mathbf{V} = d\mathbf{R}/dt$, $\mathbf{v_{12}} = d\mathbf{r_{12}}/dt$, M(the total mass) = M_1+M_2, and μ (the reduced mass) = $M_1M_2/(M_1+M_2)$. Equation 2.9 implies the following equations of motion:

$$Md^2\mathbf{R}/dt^2 = 0 \tag{2.10}$$

$$\mu.d^2\mathbf{r}_{12}/dt^2 = -\partial U(r_{12})/\partial\mathbf{r}_{12} = Z_1Z_2e^2\mathbf{r}_{12}/(4\pi\varepsilon_0 r_{12}^3) \tag{2.11}$$

According to equation 2.10, the COM will be in uniform motion throughout the collision, giving rise (equation 2.9) to a constant associated kinetic energy of ½ $M\mathbf{V}^2$. At the start of the collision, the interaction term is negligible (r_{12} is large), and the target has zero velocity ($\mathbf{v_{(0)12}} = 0$), so equations 2.6 and 2.9 can be expressed as:

$$½\ M\mathbf{V_{(0)}}^2 + ½\ \mu\mathbf{v_{(0)12}}^2 = E_{com} + E_{12} = ½\ M_1\mathbf{v_{(0)1}}^2 = E_0, \tag{2.12}$$

where E_{com} is the constant kinetic energy associated with the COM motion; E_{12} (the *relative energy*, or *energy in the COM frame*), which also constant, is the energy available for the pseudo 1-body scattering problem expressed by equations 2.9. The important point to appreciate is that the distance of closest approach (and other features of the collision) depends on the relative energy, E_{12}, rather than the Lab incident energy, E_0. By using the relation:

$$½\ M\mathbf{V_{(0)}}^2 + ½\ \mu\mathbf{v_{(0)12}}^2 = ½\ M_1\mathbf{v_{(0)1}}^2, \tag{2.13}$$

it is not difficult to show that:

$$E_{12} = M_2/(M_1+M_2)E_0. \tag{2.14}$$

Evidently, E_{12} approaches E_0 as the target mass increases relative to the projectile mass. Equation 2.9 can be expressed as:

$$½\ \mu\mathbf{v_{12}}^2 + Z_1Z_2e^2/(4\pi\varepsilon_0 r_{12}) = E_{12} = M_2/(M_1+M_2)E_0. \tag{2.15}$$

Example. A 1 MeV proton is projected at a stationary alpha particle (He nucleus). What is the distance of closest approach (r_{min}) in a direct impact collision? Would it be the same if the alpha particle were projected towards the proton with the same initial energy?

Solution. The distance of closest approach is reached when all of the relative kinetic energy E_{12} is converted to potential energy. Hence, the first term in equation 2.15 is zero, giving: $Z_1 Z_2 e^2 / (4\pi\varepsilon_0 r_{12}) = M_2 / (M_1+M_2) E_0$. Using $Z_1 = 1$, $Z_2 = 2$, $e = 1.6x10-19$ C, $M_2 / (M_1+M_2) = 0.8$, $\varepsilon_0 = 8.85x10^{-12}$ F m^{-1}, $E_0 = 10^6 e$ J, we obtain $r_{12} = 3.6x10^{-15}$ m. For the case of an alpha particle projectile, the $M_2 / (M_1+M_2)$ term is now 0.2, and the distance of closest approach is accordingly four times larger ($1.4x10^{-14}$ m).

Eqn. (2.11) is formally equivalent to the equation of motion of a particle of mass μ moving under the influence of a Coulomb potential originating from a fixed centre. The equation of motion 2.11 can be solved analytically. The Cartesian coordinate system is converted to a circular system: $(x_{12}, y_{12}) \rightarrow (r_{12}, \theta)$. Standard texts show that the scattering angle in this coordinate system (the COM system) can be expressed as:[9]

$$\theta = \pi - 2 \cdot \int_{r\,\min}^{\infty} \frac{b \cdot dr_{12}}{r_{12}\sqrt{(1 - U(r_{12}) / E_{12}) \cdot r_{12}{}^2 - b^2}} \tag{2.16}$$

where $U(r_{12}) = Z_1 Z_2 e^2 / (4\pi\varepsilon_0 r_{12})$. This integral can be evaluated for the Coulomb potential, as well as a few others such as the inverse square potential (the latter is quite useful for testing simulation programs because it represents a screened Coulomb potential).

For the Coulomb potential, $V(r_{12}) = A / r_{12}$, equation 2.16 evaluates as:

$$\theta^* = \pi - 2\arctan(2bE_{12} / A), \tag{2.17}$$

where the asterisk (*) reminds us that the angle in question refers to the COM coordinate system. The corresponding result for the inverse square potential, $V(r_{12}) = A / r_{12}^2$ is:[4]

$$\theta^* = \pi[1 - 1 / \sqrt{(1 + A / b^2 E_{12})}] \tag{2.18}$$

The COM result, equation 2.17, can be expressed in the Lab frame by means of transformations shown in the standard texts on mechanics, and in a similar fashion, scattering cross-sections for the laboratory system can be derived. These are not particularly interesting for the present discussion, so they are passed over here.

However, it is important to emphasise that knowledge of the Lab scattering angle is not enough to specify the particles' trajectories in the Lab frame, although it does specify their asymptotic Lab directions of motion. To know the trajectory, one would have to solve for the variation of $\mathbf{r_{12}}$ with time, and express this motion in the coordinates of the Lab frame. This is quite an involved problem, which also requires the evaluation of the so-called time integral, which gives the interval t between the initial and final limits of the motion ($r1$, $r2$ respectively):

$$t = \int_{r1}^{r2} \frac{dr}{\sqrt{(2/\mu)\cdot[E - U(r_{12})] - M^2/\mu^2 r_{12}{}^2}} \tag{2.19}$$

For the Coulomb potential, this integral diverges as the initial radial separation increases, which means that the ideal case of a projectile 'approaching from infinity' cannot be evaluated. Computer simulations of atomic collisions based on the binary collision approximation (BCA) model do make use of the time integral (in conjunction with a screened Coulomb interaction). The reader is referred to the research literature for further discussion of this topic.[2, 3, 15] In simulations of atomic scattering in solids, it is important to be able to track the actual motion of the projectile (not just its asymptotes) because of the need to calculate the impact parameter for a possible subsequent collision with another target atom.

2.3.4. Other Potentials

The integral in equation 2.16 can only be evaluated analytically for a few potentials. A number of approximate analytical treatments have been developed in response to this. The *small-angle* (or *impulse*) *approximation* introduces a number of assumptions which simplify the scattering integral, but it is not applicable to hard (small impact parameter) collisions.[4] Another approach is to develop universal formulae based on regression fits to parametric models (so-called 'magic' formulae).[16] Approximate techniques of this kind have an important role to play in simulation models based on the binary collision approximation (BCA), where a multiple scattering process is represented as a sequence of binary collisions. The BCA model is the basis of some well-known computer simulation programs (TRIM, MARLOWE). The main differences between the CD and BCA programs are that (a) the BCA programs neglect interactions between target atoms; (b) the BCA programs rely on algorithmic prescriptions for dealing with *quasi*-simultaneous scattering configurations. The BCA model will not be discussed further in this article: see the books by Smith *et al.* or Eckstein for compact reviews of the model,[2, 3] or the article by Robinson for a comprehensive review.[7]

The 'composite' interaction potentials used in classical dynamics simulations of projectile-solid collisions are described in chapter 4.

2.3.4. Many-Body Systems

More interesting (from the viewpoint of the surface analyst) is a system in which a projectile is scattered by two fixed centres, and where the projectile-target interaction is governed by screened Coulomb potentials: no analytic solution is possible, but the theoretical analysis of this system by approximate methods remains a subject of current surface science research.[17]

In the normal conception of a many-particle system, the scattering centres (target particles) would undergo displacement as a result of interaction with each other as well as with the projectile. Generally in this article it will be assumed that all interactions can be expressed using pairwise potentials. An N-body system accordingly gives rise to ½N(N-1) pairwise interactions which govern motion in the system. The motion of the *i*th particle in a many-particle system is then determined by the resultant of all of the forces acting upon it, which can be expressed in the following vector equation ($m\mathbf{a}_i = \mathbf{F}_i$):

$$M_i d^2\mathbf{r_i} / dt^2 = -\Sigma\ (\partial U(|\mathbf{r_i} - \mathbf{r_n}|)/\partial \mathbf{r}_i) \quad [= -\Sigma\ \nabla_\mathbf{i} U(|\mathbf{r_i} - \mathbf{r_n}|)] \tag{2.20}$$

where the summation runs over the index *n* (*n* = 1...N, omitting the value *i*).

Here is a concrete example of the evaluation of equation 2.20, for the case when particle 1 is interacting via Coulomb forces with two others (*i* = 1, n = 2,3):[*]

$$M_1 d^2x_\mathbf{1} / dt^2 = Z_1 e^2/(4\pi\varepsilon_0).\{ Z_2(x_1-x_2)/r_{12}^3 + Z_3(x_1-x_3)/r_{13}^3 \} \tag{2.21a}$$

$$M_1 d^2y_\mathbf{1} / dt^2 = Z_1 e^2/(4\pi\varepsilon_0).\{ Z_2(y_1-y_2)/r_{12}^3 + Z_3(y_1-y_3)/r_{13}^3 \} \tag{2.21b}$$

$$M_1 d^2z_\mathbf{1} / dt^2 = Z_1 e^2/(4\pi\varepsilon_0).\{ Z_2(z_1-z_2)/r_{12}^3 + Z_3(z_1-z_3)/r_{13}^3 \} \tag{2.21c}$$

Unlike the case of the binary collision, there is no advantage in transforming Newton's equations to a non-Cartesian or COM coordinate system for many-particle systems. Equations analogous to 2.1 are the starting point for computer-based classical dynamics calculations.

---

[*] The derivatives are evaluated by means of 'chain-rules' like the following: $\partial U(r_{12})/\partial x_1 = \partial r_{12}/\partial x_1 * \partial U(r_{12})/\partial r_{12} = (x_1-x_2)/r_{12} * \partial U(r_{12})/\partial r_{12}$, where $r_{12} = ((x_1-x_2)^2+(y_1-y^2)^2 +(z_1-z_2)^2)^{½}$. Terms like $(x_1-x_2)/r_{12}$ are known as 'direction cosines'.

3. Computer Implementation

3.1. Introduction

This chapter reviews some miscellaneous ideas about the development of programs for classical dynamics (CD) simulations of atomic collisions on personal computers (PCs). The chapter is intended to provide a few practical tips for research students who are commencing investigations in this area, and is not necessary reading for those who are purely users of the *Simulation Kit,* who may omit this chapter entirely.

When I first wrote this chapter, I had not seen the book by Smith et al.[2] which covers the same ground, but far more competently and thoroughly. For that reason, I have resisted the temptation to enlarge this chapter.

It is worth remarking that the development of PC based simulation programs is only possible because of the rapid increase in CPU speeds which have achieved seen since 1990 or so. Robinson mentions that a CDC 7600 mainframe in ~1980 required about 1 s for each integration timestep involving a ~1000 atom system, which is similar to the performance of Snook 2.x running on a Pentium PC in 1997.[7] However, it must be emphasised that a fast CPU cannot compensate for a poorly implemented simulation algorithm, which can increase computation time by several orders of magnitude.

A first attempt at writing a simulation program will usually produce an inefficient result; but if it runs correctly, the programmer should be encouraged, rather than discouraged, by his creation.

3.2. Computer languages and tools

Traditionally, CD programs have mostly been developed in Fortran, or to a lesser extent in C. (Examples of such programs can be found at many Internet web sites.) There are good reasons (e.g. portability) for continuing to work in these languages when command-line mainframe programs are being developed. For a PC based application, these languages can also be used, but they suffer the drawback that they are not particularly well-supported by the mainstream compiler manufacturers, and so lack the sophisticated ‘rapid application development’ (RAD) features for visual development associated with C++ and Pascal products.

Visual programming and user interface development is not a normal feature of most computer courses for scientists. It is probably wise to write early simulation routines as simple command-line Dos applications. Success in this task will increase motivation to tackle the visual programming hurdle. This is particularly important if you plan to pass on the fruits of your labours to others. Be warned that today’s PC users may simply ignore a scientific program that lacks a menu-driven user interface! Fortunately, with a compiler product like Delphi or C++ Builder it is almost child’s play to develop a minimal interface. (Version 2.2 of the *Simulation Kit* was built and compiled using Delphi 3.0).

C++ and Pascal (in most commercial implementations) are object-oriented languages which take most of the labour out of designing windows, menus, dialog boxes and other visual objects. Although it is not difficult to link together object files built from different languages, most programmers would presumably prefer to develop and debug their programs from within a single, integrated environment. (For C, this can of course be achieved with a C++ compiler.)

The prominent names in compiler technology today are Watcom, Microsoft and Borland. Modern compiler products feature a high degree of integration between the code editor, the compiler, and various supporting tools (including, hopefully, a *debugger* and *profiler*). For scientific applications, it is very important to have (a) the capability of inserting assembly language statements into the body of high-level language code; (b) a debugger which allows you to view the registers of the numeric coprocessor. (You may not use assembly language now, but you may eventually.) You may have to buy a tool for (b) separately (if you have Delphi 2 or 3, for example, you will need Turbo Debugger 32).

If you plan to distribute your program, you may need to develop a context-sensitive, online Help system for it. For Windows environments, the use of a *help-authoring tool* is recommended for this task. You can find shareware products in the `\winhelp` directories of Simtelnet's win95 and win31 collections.[*]

The final class of tools I want to mention comes under the description of *memory checkers* for instance HeapAgent (`http://www.microquill.com`) and BoundsChecker (`http://www.numega.com`). These are fantastically expensive tools, but you can download and use them for a limited period to run over your final program, which they will check for various kinds of memory and resource allocation errors.

3.3. Performance issues

It is meaningless to argue along lines such as 'Fortran is faster than C++': the relative performances of two programs depend on how the compiler manufacturers have chosen to implement floating point operations at the assembly language level. The relative performances can also vary according to the hardware, operating environment and compilation options. Borland Pascal 7, for example, used a very slow implementation of the `exp()` function, but the same function in its 32 bit successor (Delphi 2) is as tight and fast as the coprocessor architecture allows. Essentially, performance has to be determined by empirical investigation.

A code profiler (such as Turbo Profiler) may be shipped with your compiler, though these seem to be rare and expensive for the Windows 95 environment. This type of tool allows you to determine the 'hot-spots' or 'time-eaters' of your program, and thereby develop strategies for performance optimisation. For example, a CD simulation program spends most of its time either computing the Morse forces, or building the neighbour lists. You can use a profiler to identify the critical statements which you need to (a) rewrite in a high level language or in assembly language, or (b) replace by a faster algorithm (the preferred method).

Program precision is likewise dependent on low-level language and hardware details, rather than the choice of high-level language. The double-precision IEEE floating point types used by Fortran, C, C++ and Pascal are identical. However, hardware (numeric processor) performance varies from manufacturer to manufacturer.

3.4. Programming tips

3.4.1. Programming style

From the outset, you should write your code in the expectation that it will have to be modified many times. This means choosing meaningful variable names, and

---

[*] http://www.simtel.net/pub/simtelnet/

documenting the logic of obscure algorithms (Fortran and C users take note!). As far as possible, you should compile and run the program as soon as each new function / subroutine is completed: this may involve writing some temporary skeleton code which calls the new function for testing purposes. (The ability of PC compilers to integrate editing / compilation / debugging operations is a major advantage over mainframe development environments.) In this respect, the fast compilation time of Pascal is a major advantage over C++.

3.4.2. Operating systems

Since PC operating systems and compilers change so quickly nowadays, it is essential to separate the code for your user interface (‘front end’) from that for your simulation routines (‘back end’ or ‘engine’). For example, the simulation engine used by `Snook` consists of one big function (subroutine) contained in a module of its own which is called by the main program:

```
...
Simulate(TrgFile, PrjFile,...);
...
```

The same function can be used with minor modifications by a Dos program or by a Windows 95 program. Once the simulation is running, the main difference between these two platforms is in the way user inputs (e.g. Ctrl-Break) and screen outputs (e.g. messages) are handled. Instead of executing your chosen input / output method directly, you should create a procedure for the purpose:

```
...
Msg := ‘Output information’;
WriteScreen(Msg)
...
```

Now when you change OS platforms, only the `WriteScreen()` routine has to be changed.

3.4.3. User input

Input data should be validated as much as possible at the time of entry (in the *Simulation Kit*, most data validation is handled by `Spider`). Modern visual development environments provide dialog box ‘objects’ which can automate a wide variety of data validation tasks (e.g. validating the range and format of floating point and integer numbers).

Avoid ‘hard-coding’ data into your simulation routines as much as possible. It is tempting to do this when you are the sole user of the program, because you can always change the data and recompile the program as you wish. However, in the author’s experience, this procedure tends to introduce errors. If you find yourself changing hard coded constants, it is better to provide some method of inputting them into the program from an external source.

Ideally, input should be in the form of data stored in files, rather than data entered via the keyboard. File-based data mean less work for the user, and are self-archiving. However, for small programs (such as the `cone` utility) it may be too pedantic to insist on file-based inputs.

3.4.4. Dynamic memory issues

You can write small simulation programs without using pointers and dynamic memory allocation. But eventually you may wish to do so. In a Dos / Win 3 environment, one quickly exhausts the 64 k array size limitation, which applies regardless of your machine's physical RAM. Under Windows 95 / NT, more memory is available, but it has to be shared with other programs.

In general, dynamic memory allocation and deallocation is an area of programming which always requires special attention:

(a) Failure to deallocate memory will cause memory 'leaks'. The easiest way to check for these is to continuously monitor memory usage. You can see that both `Snook` 1.2 and 2.x offer options of this kind, in the form of visual memory meters. These are used by the author for debugging purposes. After running a simulation, the memory allocation should return to its original value. For a variety of reasons, such as pointer management overheads, this does not usually occur precisely. However, apparent leaks due to legitimate overheads will saturate after the routine has been run 2 or 3 times, unlike true leaks which will continue to deplete the memory pool *ad infinitum*.

(b) Failure to allocate sufficient memory (e.g. for arrays whose length is set at run-time) is another source of dangerous bugs, particularly the well-known 'out-by-one' error that plagues C programmers. Such bugs (memory overwrites) will not necessarily be indicated (by a general protection fault) even in a protected mode environment, so it is good practice to examine the contents of arrays (or at least, their first and last elements) in the 'Watch' window of a debugger as you step through the program. Once things are running, it is a good idea to temporarily modify the program so that that you are deliberately under-allocating memory, in order to see what behaviour this produces.

(c) Most non-trivial program errors are associated with illegal pointer operations. To keep these to a minimum, it is a good idea (a) to assign pointers to `nil` (or `null`) both on program start-up, and after deallocating dynamic memory; (b) to allocate or deallocate memory for a given pointer in only one place in the program. The definition of a pointer error depends in part on the operating system. Under Windows 95 an attempt to *read* unallocated array elements may generate an exception, whereas this is legal under Dos.

3.4.5. Compiler bugs

Know your compiler well: most libraries and compilers released by compiler vendors contain dozens of bugs. (Lists of these can be found on Internet sites and in Usenet groups `comp.lang.xxx` devoted to that compiler.) For this reason, you need to ascertain that your compiler package contains the source code for all libraries, so that you can debug them if necessary.

3.4.6. Data structures and algorithms

For classical dynamics simulations, the main data structures you will use are the 1- and 2-dimensional arrays. 1-dimensional arrays are used to hold vector components (x[n], y[n], z[n]), while 2-dimensional arrays are used for holding neighbour lists and possibly force components (depending on the integration algorithm you use). Do not use container classes (objects) to store vector quantities, because access to their elements is needlessly slow.

In Dos, it is useful to be able to size 2-dimensional arrays (a) beyond the 64 k barrier, and (b) to do this at runtime. A data structure which can do this is declared and implemented as follows.

```
type
  float = double;
  PFloatArray = ^TFloatArray;
  TFloatArray = array[1..8000] of float;
  PFloatMatrix = ^TFloatMatrix;
  TFloatMatrix = array[1..8000] of PFloatArray;
var
  F: PFloatMatrix;
  n, ncells, mcells: integer;
begin
  { allocate memory for an array of size ncells*mcells}
  { ncells, mcells must each be < 8000 in this example }
  ncells := 1000;
  mcells := 100;
  { total memory allocation is ncells*mcells*4 bytes }
  GetMem(F,ncells*SizeOf(PFloatMatrix));
  for n := 1 to ncells do GetMem(F^[n],mcells*SizeOf(PFloatArray));
  { initialise first cell }
  F^[1]^[1] := 0.0;
  { initialise last cell }
  F^[ncells]^[mcells] := 0.0;
  { free memory }
  for n := 1 to ncells do FreeMem(F^[n],mcells*SizeOf(PFloatArray));
  FreeMem(F,ncells*SizeOf(PFloatMatrix));
end;
```

CD simulations do not usually require an in-depth knowledge of computer science algorithms beyond what can be gleaned from numerical analysis texts like *Numerical Recipes*.[18] A possible exception is range-searching algorithms (for improving the efficiency of neighbour list construction).[2,19,20]

3.5. Program structure

In this section I want to highlight some fragments of code extracted from `Snook` for two purposes. First, the code fragments will shed light on the internal structure of the simulation routine. Also, they may be of interest for those who plan to develop their own routines. I have simplified the appearance of the code by replacing dereferenced pointers by static variables (in practice, memory for all array variables is allocated dynamically), and by removing any details of minor interest.

It should be appreciated that the bulk of the code in `Snook`'s simulation routine (i.e. excluding the user interface and event handling system) is devoted to initialisation processes, namely (a) memory allocation, (b) data input / validation, and (c) assignments to variables. The initialisations performed in (b) and (c) can be divided into two types: those that must be initialised once only (such as the potential parameters), and those that have to be initialised for every new run, i.e. every new projectile trajectory (for example, the target atom coordinates).

Although the bulk of the programming effort is devoted to the initialisation routines, the bulk of execution time is spent in computing the forces and in updating the neighbour lists. Most optimisation efforts will be focussed on these routines, and it is these sections which have to be constructed most carefully.

3.5.1. The integration loop

The code fragment shown below implements the ‘velocity form’ of the Verlet integration algorithm:

$$\mathbf{r}_{n+1} = \mathbf{r}_n + \mathbf{v}_n\Delta t + \tfrac{1}{2}\,\mathbf{F}_n\Delta t^2/\,m \tag{3.1}$$

$$\mathbf{v}_{n+1} = \mathbf{v}_n + \tfrac{1}{2}\,(\mathbf{F}_{n+1} + \mathbf{F}_n)\Delta t\,/\,m \tag{3.2}$$

On entry to the procedure, some useful composite constants are prepared, and the arrays, which on exit hold the force components (from the current and preceding timesteps), are initialised. The forces are summed by calling the `GetForce()` procedure, and finally the particle velocities are updated.

```
Procedure Get_rv_Verlet(NoOfAtoms:longint;ti:float);
var
  i,j,k: longint;
  Fx, Fy, Fz: float; // used to hold forces
  ti2div2,tidiv2,CutOff2: float; // composites
  kr,kv: float; // Composites
  ArraySize: longint;
begin                    // Get_RV_Verlet
  ti2div2 := 0.5*sqr(ti); // ti is the timestep
  tidiv2 := 0.5*ti;
  CutOff2 := sqr(CutOffDistance);

  for i := 0 to NoOfAtoms do // FNew gets value of Force from prev. step
    begin
      FNewX[i] := FOldX[i];
      FNewY[i] := FOldY[i];
      FNewZ[i] := FOldZ[i];
    end;
  ArraySize := (NoOfAtoms+1)*SizeOf(float);
  FillChar(FOldX,ArraySize,0);  // zero out all elements
  FillChar(FOldY,ArraySize,0);
  FillChar(FOldZ,ArraySize,0);

  // Calculate predictor for r
   for i := 0 to NoOfAtoms do    // Calculate drift during dt
    begin
      kr := ti2div2/mass[i]; // = ti^2/2m
      x[i] := x[i] + vx[i]*ti + FNewX[i]*kr;
      y[i] := y[i] + vy[i]*ti + FNewY[i]*kr;
      z[i] := z[i] + vz[i]*ti + FNewZ[i]*kr;
    end;

  // Evaluate forces at new r
  for i := 0 to NoOfAtoms do
    begin
      k := 0;
      j := partner[i][k];  // j is partner #k of i
      while  (j <> EndOfList) do   // EndOfList is end of partner list
        begin   // j is always > i in the partner array
          if (sqr(y[i]-y[j])+sqr(x[i]-x[j])+sqr(z[i]-z[j]) <= CutOff2) then
            begin
              Get_Force           // at r[n+1]
              (Fx,Fy,Fz, // <-- outputs these force components
               x[i],x[j],  // interaction of #i with its k_th partner
               y[i],y[j],  // which is particle #j
               z[i],z[j],
               i,j);
               FOldX[i] := FOldX[i] + Fx;
               FOldY[i] := FOldY[i] + Fy;
               FOldZ[i] := FOldZ[i] + Fz;
               FOldX[j] := FOldX[j] - Fx;
               FOldY[j] := FOldY[j] - Fy;
               FOldZ[j] := FOldZ[j] - Fz;
            end;
          inc(k);
          j := partner[i][k] // Break when j = EndOfList
        end;     // while j
    end; // i

  for i := 0 to NoOfAtoms  do  // update velocity
    begin
      kv := tidiv2/mass[i];
      vx[i] := vx[i] + (FNewX[i] + FOldX[i])*kv;
      vy[i] := vy[i] + (FNewY[i] + FOldY[i])*kv;
      vz[i] := vz[i] + (FNewZ[i] + FOldZ[i])*kv;
    end;     // for i
end; // Get_rv_Verlet
```

Each atom, #N, has a list of potential neighbours or collision partners which is stored in the partner array (the ‘neighbour lists’). Thus, the partners of atom #N are stored as integer indexes as the array elements `partner[N][0]`, `partner[N][1]`, `partner[N][2]` and so on, until a negative index signals the end of the list for atom #N. This arrangement means that time is not wasted in checking the location of all atoms in the lattice. The neighbour lists are updated periodically. To speed things further, the only atoms included in the neighbour list of an atom of index #N are those whose indices are greater than N. This means, for example, that F_{31} (the force acting on particle #3 due to interaction with #1) is computed at the time that F_{13} is computed, exploiting the fact that $F_{13} = -F_{31}$.

The `GetForce` routine returns the force components calculated from the particle vectors. Indices (i, j) identifying the particles are also passed to the routine. Based on these indices, the routine can decide whether the force should be calculated from the projectile-target or target-target potential parameters respectively.

3.5.2. The timestepping loop

The main loop of the program has a very simple structure. As shown in the simplified fragment below, this loop repeatedly calls the integration algorithm discussed in the preceding section. The loop also has responsibility for initiating rebuilds of the neighbour lists (see next section), and for updating the timer (time elapsed), the timestep and the interaction sheath (‘test range’) respectively.

```
// Perform various initialisations of variables (deleted)
// Start main loop
  Repeat

    if (NeighbourListTimer = UpdateTime) then
       begin
         CorrectTestRange; // Shld be called before GetNeighbours
         GetNeighbours(NoOfAtoms);    // build neighbour list
         NeighbourListTimer := 1;     // reset counter
         if UserAborted then break;   // break from Repeat..until
       end
    else inc(NeighbourListTimer);     // increase counter

    Get_rv_Verlet(Lattice,NoOfAtoms,ti)


    TimeElapsed := TimeElapsed + ti;  // update elapsed time counter
    inc(TimeStepsExecuted);           // update timesteps counter
    CorrectTimestep;                  // Update ti, V0 (fastest atom velocity)
```

```
Until TerminationConditionsMet;       // time & energy conditions
```

3.5.3. The neighbour lists

The routine below shows how neighbour lists are built up in `Snook` by the ‘brute force’ method. For a given atom *i*, the algorithm prepares a list of those atoms *j*,*k*,*l*... which lie within a certain range. Some work is saved by using a projective technique to filter out particles which are found to be out of range along a particular axis. However, for large lattices, this is not the fastest way to build up the lists. `Snook` offers a second option for building the neighbour lists (the ‘cell index' or 'box' method), which will not be presented here, as its algorithmic details are rather tedious.[2, 20]

```
Procedure GetNeighbours(NoOfAtoms:longint);
var
  i,j,k: integer;
  range:float;
begin
  // User key-presses/mouse clicks are processed here (not shown)

  range := sqrt(TestRange2);  // range = width of interaction sheath
  for i := 0 to NoOfAtoms do  // i must run over ALL particles
    begin
      k := 0;
      Partner[i][0] := EndOfList;   // Partner = i,j,k,..EndOfList)
      for j := i+1 to NoOfAtoms do    // Ignore j <= i
        if (abs(x[i]-x[j]) > range) // This condition filters out
        or (abs(y[i]-y[j]) > range) // the majority of atoms in the lattice
        or (abs(z[i]-z[j]) > range) then  // do nothing
        else if (sqr(x[i]-x[j])+sqr(y[i]-y[j])+sqr(z[i]-z[j]) <
TestRange2)then
            begin
              Partner[i][k] := j;
              inc(k);
              if (k = NoOfPartners) then  // Stop! (not shown)
            end;           // for j
          Partner[i][k] := EndOfData;
    end;       // for
end;           // GetNeighbours
```

3.6. Program validation

It is difficult to formally prove the correctness of a classical dynamics simulation program. The program should, of course, conserve energy, and linear and angular momentum (provided inelastic energy loss effects are not included in the simulation).

Program validation should begin with a study of the kinematics and dynamics of the binary collision, for which it is easy to calculate (for example) the correct apsidal distance by hand (see the 'Gadgets' menu in `Spider`). Three-body systems in symmetric or linear configurations can also be analysed in a similar fashion. The `cone` program, which ships with the Simulation Kit, is a useful benchmark for the correctness of binary collision simulations (`cone` uses the slow but accurate Runge-Kutta algorithm, and was programmed quite independently of the *Simulation Kit*).[11]

There are relatively few benchmarks available in the literature which serve to test program performance and accuracy under realistic running conditions. Gärtner *et al.* recently reported the results of a round robin comparison of 6 established classical dynamics (CD) programs.[21] The various programs were used to calculate the elastic energy loss and angular deflection suffered by a projectile passing through a thin crystalline target. (The reader is referred to the original source for specific details of the simulation problems.) The results from the round robin are compared with the corresponding results calculated using Snook in Table 3.1. The agreement between Snook and the round robin results is generally satisfactory, although two of the angular deflections fall just outside the range reported by the contributors to the round robin study.

Table 3.1. Calculation by Snook of the average elastic energy losses (ΔE) and root mean square angular deflections ($\Delta\theta$) suffered by projectiles passing through thin crystalline targets, compared with the results of a round robin study.[21]

System	ΔE (eV)	ΔE (eV), ref. [21]	$\Delta\theta$ (°)	$\Delta\theta$ (°), ref. [21]
0.2 keV B-Si(100)	66.1	63.1 - 68.4	35.2	32.6 - 34.5
0.5 keV B-Si(100)	64.6	55.9 - 65.1	25.7	22.1 - 25.1
1.0 keV Ar-Cu(100)	424.9	412.1 - 428.4	19.6	19.1 - 19.8

Nevertheless, it is still puzzling to observe the relatively large discrepancies which exist between different MD programs, e.g., the variation of 9eV in the ΔE values reported for 0.5 keV B-Si(100) (Table 3.1). Such discrepancies are far greater than the typical average energy conservation errors expected from the integration algorithm (~ 1 eV for a 0.5 keV system). Probably the only way to understand discrepancies at this level is for the authors of different programs to compare the results of integrating individual trajectories.[*]

---

[*] Some evidence that the discrepancy is not due to integration errors is furnished by the observation that the different algorithms available in Snook give rise to practically identical results.

4. The Physical Model

4.1 Introduction

This chapter will review the physical model and assumptions underlying the simulations performed by the *Simulation Kit* program `Snook`. All simulation models are founded upon a multitude of assumptions. To use and interpret the results from a simulation, one must understand the limitations of the simulation as well as its capabilities.

It is extremely difficult to prove the physical (as opposed to computational) correctness of any particular simulation model. The observation of individual projectile trajectories or the ensuing collision cascades is not experimentally feasible, so only average quantities derived from simulations can be checked empirically. Effectively, the information generated by simulations for a 6N-dimension phase space is mapped onto a 6-dimension space for purposes of experimental comparison. Any simulation which produces an experimentally-corroborated *mean* distribution of particles in phase space will thus appear to correctly represent the physical system, even if it is correct “by accident.”[*] Furthermore, many experimental scattering techniques sample a phase space of limited dimensionality: e.g. ion scattering spectroscopy samples only p_x and p_z (and possibly p_y), and therefore only tests a restricted number of the simulated phase space averages.

As it executes a simulation, `Snook` writes real-time output to the screen in a manner which reflects the user's preferences (i.e. degree of verbosity). The meaning of this output should be broadly clear, but it is explained in the tutorial file, which ships with the *Simulation Kit* (`c\sk22\tutorial\tutorial.doc`). The collision dynamics can also be viewed graphically. Verbose output and graphical views are extremely useful when fine-tuning a simulation project. During a production run, however, you should switch these options off, and 'minimise' the `Snook` main window, in order to obtain maximum performance.

In the following brief review, I have tried to address the issues which I think will be most important for an understanding of the design of the *Simulation Kit*. I hope that readers will point out omissions to me. I have not attempted to discuss competing classical dynamics simulation models (e.g. those incorporating many-body potentials).

4.2. Initial conditions

4.2.1. Lattice structure

The lattice structure used by `Snook` is derived from the Target file (*.`TRG`) which is a listing of coordinates and other information about the particles in the target. The *xy* plane of the coordinate system is parallel to the surface, while the *z*-axis corresponds to the surface normal. The negative *z*-direction corresponds to movement into the lattice. The projectile approaches the surface in the *xz* plane if the azimuthal angle is set to zero (in the Run file), starting from the +*x* direction, and moves in the negative *x*-direction towards the location of the *anchor atom* (see below).

If thermal displacements are selected (in the Model file), then at run-time `Snook` will add appropriate (random, Gaussian, Debeye-Waller) thermal displacements, including

---

[*] One can draw a rough analogy here with the ambiguity which sometimes arises in interpreting LEED patterns purely on the basis of the kinematic theory.

any surface effects specified by the user. In this case, the user also has an option (under `Snook`'s Options | Simulation menu) to add an average kinetic energy of 3/2kT to the lattice atoms (which is derived from a Maxwellian distribution of velocities). If this option is not selected, the lattice atoms are stationary at the start of the simulation (the normal case for keV scattering). Chapter 8 discusses the implementation of thermal vibration effects in more detail.

4.2.2. Projectile coordinates

The starting position of the projectile is referenced to the coordinates of an *anchor atom* in the target (see fig. 4.1): these are the coordinates listed in the first line of the Target file (i.e. the coordinates prior to application of thermal displacements). For small impact parameters, the anchor atom is the projectile's first collision partner. By default, in all lattices generated by `Spider`, the anchor atom is located at the origin (0,0,0) of the coordinate system.* The vertical projectile position relative to the anchor atom is determined by the *z0* parameter (normally 3 Å) in the Impact file. The (*x*, *y*) coordinates of the projectile are determined (a) by the impact parameters (along the *x*- and *y*-directions) for the current run (trajectory), which are listed in the Impact file; (b) by the projectile altitudinal and azimuthal angles of incidence.‡ The angles in (b) also determine the relative magnitudes of the projectile starting velocity components.

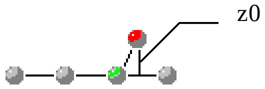

Figure 4.1. Starting geometry for a simulation run. The uppermost particle (red) is the projectile, which is located at a vertical distance *z0* above the anchor atom (green, third from the left) in the target lattice. The lateral displacement of the projectile relative to the anchor atom is determined by (a) the impact parameter for that run, (b) the projectile angles of incidence. See the `Spider` on-line Help for more information on this subject.

The arrangement just described has the advantage that it allows the same set of input files to be used for similar scattering systems with different altitudinal directions of projectile incidence (with the sole difference that the particular altitudinal angle involved has to be specified in the Run file).

4.2.3. Particle masses

The isotopic distributions of the target atoms are not known by the program (`Spider`) which generates the Target file (the target particles are assigned the user-specified

---

* If you rotate the lattice in the *yz* or *zx* planes you will normally have to edit (cut/paste) the resulting Target file to ensure that your desired anchor atom coordinates are found in the first line of the new file.

‡ The projectile altitudinal angle of incidence is a "polar" angle which takes values between +90° (normal to surface) and 0° (parallel to surface). The projectile azimuthal angle of incidence is the angle subtended around the z-axis in the xy plane; it is zero for incidence parallel to the x-axis, and 90° for incidence parallel to the y axis. The angular convention used by `Winnow` (and `Spider`'s "User-Defined" Run file output option) uses (unlike here) a strict sign convention based on the direction (sign) of vector components. You should consult the `Winnow` on-line Help for details.

average mass), but it is possible (although tedious) to modify the atomic mass data in the Target file by manual editing of each entry. The projectile mass is likewise fixed at the value stipulated in the Projectile file.

4.2.4. Initial conditions data

If you want to examine the state variables (position, momentum and others) which were actually applied at the start of the simulation, there is an option in `Snook` which causes them to be dumped to a file `inivars.snk`, from which they can be regenerated by `Winnow`'s Process | Convert command.

4.2.5. Reduced impact zone

A simulation project typically consists of a large number (10^2 - 10^5) of simulations of independent projectile trajectories, often referred to as ‘runs’ in the *Simulation Kit* documentation. The trajectories are directed towards a representative region of the surface of the target lattice which the author calls the *reduced impact zone*. Other terms are in use in the literature.[2]

The reduced impact zone is an area of the periodic surface sized in such a way that every possible incident trajectory on the surface can be mapped to a symmetrically equivalent trajectory directed into that zone. The dimensions of the reduced impact zone depend on the surface orientation, and the symmetry inherent in the collision problem (i.e. the direction of projectile incidence). Fig. 4.2 illustrates the reduced impact zone for a FCC (111) crystal surface. The reduced impact zone is always smallest at normal projectile incidence.

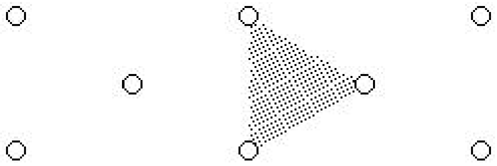

Figure 4.2. Projection of the “representative region” or “reduced impact zone” for projectiles normally incident onto a FCC (111) surface. The atoms at the corners of the zone lie in 3 different layers of the target. (These data are taken from the TRG and IMP files found in the `\examples\ne-ag111` directory of the *Simulation Kit* installation.)

The Impact file of a `Snook` simulation project contains the impact coordinates which collectively define the dimensions of the reduced impact zone. The `Spider` online Help includes an extensive discussion on the topic of reduced impact zones, and includes templates for the most common surfaces.

For an *ideal* lattice, there is no error introduced by using a reduced impact zone which is, for example twice or four times as large as the true zone. However, this can be inefficient in computational terms, as it may introduce redundancies in the output data.[*]

---

[*] However, even under these circumstances, redundancies do not generally occur in the Impact files generated by `Spider`, because of the character of the underlying generating algorithm (see `Spider` Help for more on this topic). Here is a simple numeric illustration. For 5 impacts along an axis of length a, the impact points generated by `Spider` will be 0, $a/5$, $2a/5$, $3a/5$, $4a/5$ while for 5 impacts along an zone axis of length $2a$ the impact points will be: 0, $2a/5$, $4a/5$, $6a/5$, $8a/5$;

The problem of collision containment also encourages the use of a restricted impact zone.

The introduction of randomly-applied thermal vibration effects has two consequences in connection with the impact zone; (a) each target configuration is unique when vibrational effects are included in the simulation, so every simulation will yield different results; (b) the thermal displacements blur the notion of symmetrically equivalent trajectories (indeed, the same projectile starting coordinates will generally give rise to different collision cascades, because of the different displacements of the target atoms).

It is sometimes convenient to use an impact zone that is larger (by an integral factor) than the reduced impact zone. A typical example occurs during the simulation of processes as a function of projectile altitudinal angle, where one may prefer (for the sake of simplicity) to use the same impact zone (i.e. Impact file) at normal incidence as was used for the simulations at oblique angles of incidence. (The preceding paragraph and its footnote explain why even this case will be free of redundancies in simulations that include vibrational effects.)

4.3. Interaction potentials

4.3.1. Interaction Model

Atomic interactions modelled in the *Simulation Kit* are based on analytic pair-potential functions as specified in the Model file.

The *Simulation Kit* user is required to select the type of potential employed by the simulations. There are two broad choices: (a) screened Coulombic potential; (b) composite potential. Potential (a) always applies to the projectile-target interaction. The user may select either (a) or (b) for target-target interactions. In both cases the potential is cut-off at a finite distance specified by the user (normally between the 1st and 2nd nearest neighbours, or between the 2nd and 3rd nearest neighbours).

The Morse and screened Coulombic potentials, although fairly well established, are not the only potentials being used for contemporary simulation research, nor are they the most realistic. Their main advantage lies in their suitability for a “universal” approach to simulation, such as that adopted by the *Simulation Kit*.[*] Most pair potentials are fitted to equilibrium state properties, so they presumably are most realistic for the undisturbed lattice.

Coulombic potential at small internuclear separations, a Morse potential at large internuclear separations, and a cubic spline function which connects the two in the intermediate region. The spline function is computed automatically, based on the user’s specification of the spline region limits.

---

if the lattice repeat length is a, the latter series is equivalent to: 0, $2a/5$, $4a/5$, $a/5$, $3a/5$, i.e. there is no redundancy when the impact points are folded back inside the true reduced impact zone.

[*] A future development of the Simulation Kit (mid-1998?) should include support for embedded atom potentials.

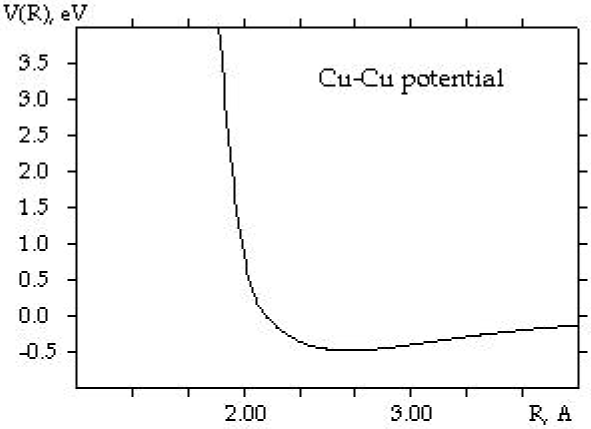

Figure 4.3. Composite potential, V(R), for Cu-Cu interaction. In the region shown, the potential consists of a cubic spline function (1.5-2.1 Å) joined to a Morse potential (R > 2.1 Å).

Fig. 4.3 shows the attractive part of the default composite Cu-Cu potential generated by `Spider`'s Model option. The corresponding screened Coulombic potential for Cu-Cu (not shown) decreases far less slowly towards zero in the region shown, and lacks any attractive potential well. The form of the attractive potential is only significant in collisions in which the apsidal distance falls outside the core region described by the screened Coulombic potential. (For Cu-Cu, for example, this corresponds to Lab collision energies below ~100 eV.) The explanation for this is that the scattering integral function (equation 2.16) is singular at the collision apsis, and so is largely determined by the force function in its vicinity. A recent round robin study demonstrated that the scattering of *projectiles* of a few hundred eV energy was independent of the form of the potential used for modelling target-target interactions.[21]

4.3.2. Screened Coulombic potential

The short range potential used by `Snook` is classified as a screened Coulombic potential. There are 3 variants in common use, which are named after their developers: the (Ziegler-Biersack-Littmark) ZBL potential, the Moliere-Firsov potential and the Moliere-Lindhard potential. The *screening length correction* is an adjustable parameter which may be used to modify the range of these potentials to improve their accuracy.

The user of the *Simulation Kit* has to select: (a) one form of screened Coulombic potential to represent the projectile atom-target atom interaction; (b) one form of screened Coulombic potential to represent the target atom-target atom interaction; (c) two 'screening length adjustment' factors which typically fall in the range 0.7-1.0 (1.0 is normal for the ZBL potential). The target-target and projectile-target interactions do not have to be of the same form or have the same screening length correction.

The analytic form of a screened Coulombic potential for interacting atoms of atomic number Z_1, Z_2 respectively, is as follows:

$$V(r) = Z_1 Z_2 e^2 . \chi(r/a)/4\pi\varepsilon_0 r \tag{4.1}$$

The screening function $\chi(r/a)$ has the properties $\chi(0) = 1$, $\chi(\infty) = 0$. It is normally represented in the general form:

$$\chi(r/a) = \Sigma_i \alpha[i].\exp(-\beta\,[i].r/a), \tag{4.2}$$

where the index *i* runs from 1-3 (Moliere potentials) or 1-4 (ZBL potential), and the $\alpha[i]$ and $\beta[i]$ are coefficients (the former satisfying $\Sigma\,\alpha[i] = 1$); *a* is the screening length (defined below).

The Moliere potentials and the ZBL potential define their respective parameters ($\alpha[i]$, $\beta[i]$ and *a)* in various different ways.[*]

The Moliere potential has two variants, according to the way in which the screening length (*a*, in A) is chosen:

$$a = 0.4685/(Z_1^{0.5}+Z_2^{0.5})^{2/3} \quad \text{(Firsov form)} \tag{4.3a}$$

$$a = 0.4685/(Z_1^{0.33}+Z_2^{0.33})^{0.5} \quad \text{(Lindhard form)} \tag{4.3b}$$

In practice this distinction is irrelevant since (judging from the literature) every user of this potential seemingly adds his or her own *screening length correction*. The screening length correction is a 'correction' factor (< 1.0) which is used to scale the screening length parameter in screened Coulombic potentials, particularly the Moliere potential. A typical value for this correction for the Moliere potential would be 0.8, usually chosen by fits to experimental data e.g. impact collision ion scattering spectrometry.

The screening length (*a*, in A) for the ZBL potential is defined as:

$$a = 0.4685/(Z_1^{0.23} + Z_2^{0.23}) \tag{4.3c}$$

For the ZBL potential, you would normally choose a value of 1.0 for the screening length correction, unless you have reason to do otherwise.

You should be wary of changing the screening length correction parameter arbitrarily, as it has enormous effects on the potential and unrealistic values could conceivably undermine the credibility of the simulations.

Which screened Coulombic potential is best? This question cannot be answered with rigour. However, if you are not going to search for an optimum screening length correction, the ZBL potential is probably the easiest choice because it is normally used in 'unadjusted form', i.e. with a screening length correction of 1.0.[#] Nevertheless, in a particular case, the Moliere potential with an optimally adjusted screening length may well be closer to reality than the unadjusted ZBL potential.

The problem is knowing what correction to apply to the Moliere screening length. The well-known MD studies of the Ar-Cu system by D.E. Harrison's group in the late 1970’s were mostly carried out using the Moliere potential (with a screening length correction of around 0.85). In early work a factor of 0.8-0.9 was typical, but more recently even values < 0.7 seem not to excite comment: e.g. 0.68 was used for 4 keV Ar-Ni in ref. [22].

---

[*] These coefficients of the screened Coulombic potentials can be found in the review articles cited in the Introduction, or in the `Spider` online Help.

[#] In Chapter 6 a case is made out for adjusting the screening length in the He-Cu ZBL interaction.

Correction factors have an enormous effect on the potential, because they involve exponentiation. In fact, the effect of the corrections may be more significant than some of the terms in the original potential!

The ZBL potential is generally used without any screening length correction, and this fixed form may be regarded as an advantage or a disadvantage, depending on how highly you rate the potential. For the Ar-Cu system at least, this author notes that the ZBL potential fits closely the *ab initio* potential calculated by Broomfield et al.[23]

Nordlund *et al.* compared the ZBL potential for C-C, Si-Si, N-Si and H-Si with Hartree-Fock (HF) potentials; [24] they found agreement typically to within ca. 3% for $V(r) < 5$ keV, and ca. 5% for $V(r) < 10$ keV. The worst agreement was for the C-C system (~5% at 3 keV).

The conclusion must be that in any serious study of an ion-surface collision one should experiment a little with the potential, to see whether this has any influence on the outcome of the simulation averages.

4.3.3. Morse and spline potential

The Morse potential V(r) is defined by:

$$V(r) = D\{ \exp{-2\alpha (r-r_0)} - 2\exp{-\alpha (r-r_0)} \} . \tag{4.4}$$

The parameters D, α and r_0 are derived from fits to bulk solid state properties,[25] and may vary somewhat from author to author, depending on the assumptions (especially the range of action) involved. This pair potential has a minimum when $r = r_0$, at which point $V(r_0) = -D$. The parameter r_0 is of the same order, but greater than, the lattice nearest neighbour distance (r_{NN}), unless the interaction is cut off at r_{NN}, in which case $r_0 = r_{NN}$.

The Morse potential, or any simple pair potential, is not particularly well suited for modelling polyatomic processes that are rigorously directional in character, e.g. covalent bonding in a material like diamond or silicon. This does not mean that such solids cannot be modelled via a Morse potential, only that effects which depend on directional bonding cannot be modelled. Such effects include equilibrium state properties such as defect formation energies.

The Morse parameters derived for a crystalline element are not the same as those applicable to a diatomic molecule of the same element. This is a recognised failing of the pair potential model.[26]

The degree of validity of the centre-to-centre (pair potential) model of interatomic forces (in the equilibrium solid) can be assessed by consideration of the elastic constants of the solid of interest. If the centre-to-centre model holds, then it can be shown for cubic crystals that the elastic constants C_{12} and C_{44} (also known as C_{xxyy} and C_{xyxy}) should be identical.[25] Here are C_{12} / C_{44} ratios for some elemental solids: Fe (1.15); Cu (1.61); Ni (1.12); Ag (2.02); Al (2.18) diamond (0.22); Si (0.8); Ge (0.72).[25,27] This comparison indicates that the centre-to-centre force model holds better according to this criterion for Si and Ge than it does for Cu, but that it breaks down badly in the case of diamond.

If you choose to include a Morse potential, `Snook` will also calculate the cubic spline function required to join the Morse potential to the screened Coulombic potential. You can preview the composite potential (screened Coulombic + spline + Morse) in `Spider`.

Note: this spline potential is recalculated by `Snook` at run-time: no explicit information about it is written to the Model file.

If you choose not to use a Morse potential, the simulations will use the screened Coulombic potential up to the range specified by the potential cut-off. This is adequate for ISS simulations, for example, because (a) the interaction time is short; (b) the process being modelled involves short-range interactions. Robinson gives a good discussion on this subject.[7]

The projectile-target potential used by `Snook` is always a screened Coulombic potential, regardless of whether or not you have selected the Morse potential option.

If the Morse potential option is selected, a cubic spline function will be used to join it to the screened Coulombic potential between the limits you specify in the Model file. The objective is to choose spline limits that give a smooth transition between the two potentials (no unphysical minima or maxima). The limits are chosen by trial and error. The composite potential will always show a smooth fit at the spline function limits. However, the force function will normally have a discontinuity in gradient at the upper and lower limits of the range covered by the spline function. The type of discontinuity to avoid is one which introduces a spurious maximum or minimum at one or both of the spline limits.

If, for some reason, you wish to employ a cubic (spline) potential all the way up to the cut-off distance, you can do so effectively by specifying a tiny separation between the high spline limit and the cut-off distance (e.g. cut-off distance = 3.8 Å, high spline limit = 3.7999 Å).

A restriction you need to be aware of is that the spline limits should both be greater than or equal to 1.0 Å. This restriction relates to the way in which the Morse look-up tables are implemented. The condition is enforced by `Spider` and checked by `Snook` at run-time.

See the `Spider` online Help for further comments about Morse parameters.

4.3.4. Surface and bulk binding energies

A surface binding energy correction would not be needed in a classical dynamics simulation if the pair interaction potential completely described the system dynamics. Many workers have argued for the inclusion of a surface binding energy term (planar surface potential or barrier) to describe the non-local, long-range interaction between an escaping atom and the free electrons of the lattice.[7] This kind of model (which is vaguely reminiscent of the pseudo-potential approach in solid state quantum theory [27]) views the atom-lattice interaction as a superposition of pair potential effects, plus effects caused by a 3-dimensional “potential well.” Trajectories are only influenced by the continuum potential when particles cross the potential barrier, e.g. in sputtering.

By contrast, Harrison argues that in a classical dynamics simulation the surface binding energy correction is unnecessary, because its influence is implicit in the Morse potential deduced from solid state properties.[5]

It is apparent that theoretical models describing "structureless media," or those which neglect interactions between target atoms (e.g. BCA programs) can benefit from the introduction of a surface binding energy term, since there are no point sources of potentials. However, it is far from easy to decide on an appropriate value for the surface binding energy term in a classical dynamics simulation which already incorporates a Morse potential. There are theoretical grounds for expecting the total surface binding energy (Morse + planar potential) to be roughly the same as the bulk cohesive energy U_0.*

If you wish to include a surface binding energy term (E_S) in your simulation, you can specify its magnitude in the Model file of your *Simulation Kit* project. In the current implementation of `Snook` (version 2.2), any required corrections to ejected particle trajectories will be made at the moment when the particle leaves the surface. This is in contrast to previous versions of the program, in which the correction was made only at the end of a simulation run. The *Simulation Kit* additionally allows you to specify (in the Model file) a 'bulk' binding energy term which is applied in a similar fashion to particles which attempt to exit the target by any of its side faces.

[Tip for advanced users: surface and bulk binding energy corrections are only applied to the motion of a particle if its `ofEmitted` and `ofNotContained` option flags are not yet set (see Chapter 10).]

The surface / bulk binding energy (E_S) correction implemented by `Snook` takes the form of reducing the appropriate velocity component of any particle (including the projectile) which is found to be emitted from the lattice, such that its kinetic energy falls by E_S. In cases where $\frac{1}{2}mv_Z^2 < E_S$, the energy deduction is not carried out: instead, the sign of the velocity component is reversed, and the particle suffers a reflection at the internal surface of the target. The definition of 'emitted' or 'not contained' in this context is that (a) the particle separation from the bounding surface is greater than the cut-off distance of the potential, and (b) the velocity component is in a direction incident on the surface. A binding energy correction is applied only once (if at all) to any given escaping particle. But a non-escaping particle may be reflected internally at the surface many times.

Thus, to sum up, the net effect of the correction is to apply either a 'reflection' or a 'refraction' to the trajectories of particles that cross the surface. Note that the implementation of this feature does not conserve linear momentum. for those particles that are reflected or refracted at the surface.

You should specify a value of 0.0 for the Surface (Bulk) Binding Energy (the default value) if you don't want to include a surface (bulk) energy correction. If you choose to include a non-zero binding energy, you will have to select a suitable value based on literature sources external to the *Simulation Kit*. None of the example projects discussed in this article incorporated such binding energy effects.

4.3.5. Image potential

A charged particle approaching the surface of a conductor experiences a classical image potential $V(r) = e^2 / 16\pi\varepsilon_0 r$, where r is the particle-surface separation.[14] If r is expressed in Å, then $V(r) = 3.6 / r$ eV. Since most primary projectiles are charged particles, there is clearly some possibility that image potential effects can influence the trajectories of slow projectiles. This would take the form of an acceleration towards the surface along the z-direction, with a corresponding increase in projectile kinetic energy (on the order of a few eV).

---

[*] The reasoning is as follows.[7] (a) The energy required to break N bonds and remove a *single* atom from the bulk into the gas phase should be roughly twice the cohesive energy, since atom formation by bulk vaporisation creates on average *two* gas phase atoms for every N broken bonds, where N is the coordination number (i.e. $E_B \sim 2U_0$); (b) the surface binding energy should be roughly half of the bulk binding energy (i.e. $Es = 0.5E_B$). Hence, $E_S \sim U_0$.

The *Simulation Kit* does not yet implement the handling of image potential effects. This subject is rarely discussed in the literature of scattering simulation, although it may be important in connection with the angular distribution of secondary ions, for example.[28] A complicating factor is the strong likelihood that charged particles will be neutralised at distances of 2-3 Å from the surface, thereby eliminating the image potential.[8] However, if you plan to run simulations involving projectiles of less than ~100 eV energy, you should at least be aware of the possibility of image potential effects, which will tend to increase the projectile velocity in the *z*-direction, and thereby increase the effective bombardment energy.

This topic will be given a better coverage in a future release of this article.

4.3.6. Many-body potentials

A number of many-body potentials have been developed for metals which describe their state properties better than pair potentials can. The Sutton-Chen potential is among them.[29] The Morse and Sutton-Chen (SC) potentials should yield similar cohesive (potential) energies for the equilibrium lattice configuration. However, the cohesive energies of a disturbed atomic configuration are anticipated to diverge with increasing lattice disorder.

Fig. 4.5 illustrates the time evolution of the total lattice cohesive energy in a single collision cascade in Cu (where the dynamics are governed by a Morse interaction), using both the Morse and SC potentials. The purpose of the figure is to compare the Morse and SC estimates of system potential energies for a realistic sequence of non-equilibrium target structures. For each configuration at a certain elapsed time, the Morse and SC cohesive energies are compared (via their ratio). In this case the cascade onset was around 20 fs, and the target became (visually) amorphous above ~300 fs. In the latter configuration, the SC and Morse estimates of system cohesive energy differ by as much as 15%. However, the discrepancy is only 5% if timescales less than 100 fs are considered. Except at the beginning of the cascade, the Morse potential gives a lower cohesive energy estimate than the SC potential.

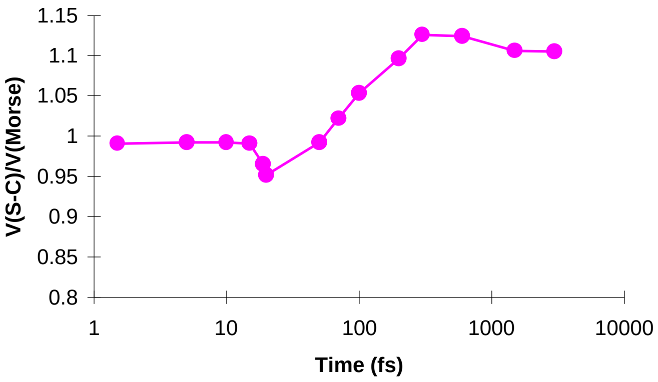

Fig. 4.5. Ratio of Sutton-Chen and Morse potentials, V(S-C) and V(Morse) respectively, for various Cu lattice configurations arising at different times in a single collision cascade (1.0 keV Ar incident on a 986 atom Cu(100) target). Note the logarithmic time scale. The Morse potential

was used to calculate the system dynamics. The total system potentials were calculated using a cut-off distance of 1.01*a*, where *a* is the Cu lattice constant. The same qualitative behaviour was observed if the Morse potential cut-off was increased to 2.01*a*.

4.4. Integration methods

The classical equations of motion are integrated by Snook using any of several finite difference integration algorithms. The Verlet algorithm, which is used by default, has already been presented in eqns. 3.1 and 3.2 of chapter 3. Another algorithm offered in Snook's simulation options is the HGE-A algorithm, which is a so-called *predictor-corrector* method:[6]

Predictors:

$$\mathbf{r}_{n+1} = \mathbf{r}_n + \mathbf{v}_n\Delta t + \mathbf{F}_n\,\Delta t^2/2m \tag{4.5a}$$

$$\mathbf{v}_{n+1} = \mathbf{v}_n + 0.5[\mathbf{F}_{n+1} + \mathbf{F}_n]\,\Delta t/2m \tag{4.5b}$$

Corrector for $\mathbf{r}_{n+1}$:

$$\mathbf{r}_{n+1} = \mathbf{r}_n + \mathbf{v}_n\Delta t + 0.25[\mathbf{F}_{n+1} + \mathbf{F}_n]\,\Delta t^2/2m \tag{4.5c}$$

The meanings of the symbols are: $\mathbf{r}_n$, $\mathbf{v}_n$, $\mathbf{F}_n$: position, velocity, total force vectors at n^{th} timestep; Δt: size of the current timestep.

These integration algorithms are uncontroversial. In general, their accuracy is checked with reference to the accuracy of energy conservation, and this is how the user should evaluate them. The accuracy of the integration depends on the timestep specified by the user in the Run file of a simulation project. One integration algorithm offered by Snook, the Beeman algorithm, should be used with a fixed timestep, and is therefore considerably more inefficient than the other choices. In general, the Verlet algorithm should prove adequate for all simulations, but it may be interesting to check one of the others for consistent behaviour from time to time.

The total system energy, as the sum of potential energy (PE) and kinetic energy (KE), can be calculated at any instant from the current particle positions (using the analytic potentials) and velocities. This can be compared with the energy calculated at the start of the simulation, which yields the energy error ΔE. Snook uses the formula $\Delta E/(KE + PE)$ to report energy conservation, but since the PE is usually negative, this formula tends to overestimate the importance of the error compared with the alternative measure: $\Delta E/(KE + |PE|)$ used by some writers. However, since ΔE is normally less than 1% the distinction is not particularly important.

The forces appearing in eqns. 4.5 are not necessarily calculated directly from the analytic potential functions. In Snook, Morse forces are calculated from a look-up table (essentially by indexing the inter-particle separation, *r*, into an array of pre-calculated values, with interpolation if necessary). Other forces (spline, screened Coulombic) are calculated directly from the respective analytic functions. The majority of particles in the system interact at any instant via Morse forces, so the use of a look-up table speeds the force calculations significantly.[*]

---

[*] Floating point calculations, especially those involving the exponential or other functions, are extremely time-consuming.

Another device which improves the speed of calculations is the use of neighbour lists. A particle's neighbour list is a list of those other particles in the system with which it may interact in the period before the next list update (normally carried out every ~10 timesteps). The neighbour lists are built up by taking into account the potential cut-off distance and the velocity of the fastest particle in the system (which determines the width of the sheath that has to be examined for potential collision partners). The 'Number of Partners' parameter which is specified in the Run file defines the maximum allowed size of the neighbour list (i.e. the memory allocation for the list). This parameter is typically set at 50-60, although larger values might be needed on occasions.

Snook does not employ the 'moving atom approximation' described in refs. [30,31], which ignores the effect of small forces (in effect introducing a displacement threshold). The only special treatment that Snook accords to small forces is in connection with the HGE-A algorithm, eqns. 4.5: small pairwise forces can be treated as if they were constant during the integration timestep, the corrector force being approximated to its value at the start of the timestep. The definition of small forces is selected by the user of Snook (in the Options | Simulation dialog).

4.5. Inelastic processes

Inelastic processes were neglected in versions 1.0 - 2.1 of the *Simulation Kit*. The physical model used in the current version is described in Chapter 9.

Inelastic energy loss processes are not well understood for keV particles, and all extant models have something of a heuristic character. Experimentally, it is not always easy to identify the effects of inelastic energy losses. The influence of discrete inelastic events on simulated processes (e.g. sputtering) has been examined in detail recently by Shapiro and Tombrello.[32,33] They discuss several methods of incorporating inelastic effects into simulation models, and find that in some cases the inclusion of the inelastic effects lead to substantial differences (~30%) in the predicted sputtering yields compared with the elastic model. At first sight, this is an unexpected result, since for 5 keV Ar bombardment of Cu(100), the average inelastic energy losses were computed to be 50-100 eV, or 1-2% of the projectile energy. These inelastic losses are of the same order of magnitude as the energy losses associated with integration errors in the simulation (~0.5%). However, the explanation probably lies in the fact that inelastic events are correlated with the same hard collisions that result in efficient sputter yields.

4.6. Binary targets

Snook is principally designed for use with elemental targets. However, it can be used with binary (or arbitrarily complex) targets provided that each of the various long-range target-target interactions (X-X, X-Y and Y-Y respectively) can reasonably be described by an *identical* Morse function. This approximation may well hold good for some common binary alloys or compounds, XY, such as CuNi or GaAs, in which the constituent atoms have similar electronic shells and occupy similar atomic sites. Snook runs noticeably more slowly with binary targets (which it detects by checking the atomic numbers in the Target file during initialisation), because these involve significantly more computations.

It should be noted that in binary targets, the calculation of the several screened Coulombic interactions (both target-target and target-projectile) will always be handled correctly, regardless of the target composition. Spline functions joining the screened

Coulombic potential to the (common) Morse potential will also be appropriate for each target-target interaction permutation (X-X, X-Y and Y-Y respectively).*

The imposition of a single Morse function on all long-range target-target interactions may disqualify the simulation model from use in some applications: for example, preferential sputtering effects in alloys presumably cannot be accurately reproduced by such a model.* However, this restriction would probably be irrelevant for the simulation of impact collision ion scattering spectra, which are mainly influenced by the short-range potential.

Chapter 10 offers some suggestions on how to simulate collisions involving systems with adsorbed overlayers, by using options flags associated with each particle to switch off or modify the form of the short-range potential. This is a partial solution which addresses the problem of target stability in such a system. However, it does not solve the problem of modelling the pair interactions.

The simulation of processes in binary targets is another area in which the author hopes to improve the *Simulation Kit* in the future. The author, as always, welcomes the views of users of the *Simulation Kit* on this matter. However, for the foreseeable future, there will be a difficulty in obtaining credible pair potential parameters for binary solids: the most promising way forward appears to be via many-body potentials.

---

* One of the main reasons why Snook runs slowly with multi-component targets is that the spline function coefficients have to be recalculated many times. It is prudent, for this reason, to define spline limits which are as narrow as possible, so that particles spend less time in the spline region.

* I am assuming here that the attractive potential is important in preferential sputtering, but I may be wrong.

5. Sputtering Simulations

5.1. Introduction

Sputtering simulations at a fixed incident angle are one of the easiest simulations to set up and perform, and they do not make unreasonable demands on computing resources. Some difficulties arise in the interpretation of results (see below), but general agreement with experimental sputtering coefficients is not difficult to achieve. More work is required for simulations of sputtering involving various projectile incident angles (see chapter 6), but no new principles are involved.

When setting up a sputtering simulation, you need to watch for problems of lattice containment and lattice stability.

5.1.1. Lattice containment

Lattice containment refers to the tendency for the effects of the collision to propagate beyond the boundaries of the finite cluster used in the simulation. If you are using `Snook` 2.1, this problem can be identified by visual inspection. If you are using `Snook` 1.2, the check for size effects is to run the simulation on differently sized lattices, and compare the resulting estimates for sputtering coefficients. For sputtering simulations involving keV projectiles you may have to tolerate some lattice containment violations, since to remove them completely may require an unrealistically large lattice.

5.1.2. Lattice stability

Lattice stability becomes an issue in sputtering simulations because of their long time span (300-500 fs normally). Over this time scale, you may see a tendency for the surface layer of the lattice to “relax” towards a configuration which is more stable from the viewpoint of classical mechanics. Relaxation effects are inherent in the assumption of a pair-potential description of the lattice.

Things that can affect this relaxation behaviour are (a) the parameters defining your Morse potential (Model file), (b) the cut-off distance used with your Morse potential, (c) the options controlling the amplitude of thermal displacements and the target temperature, (d) surface structure. A good test for lattice stability is to use a projectile of negligible kinetic energy (0.1 eV, for example) and start it far away from the surface (the starting point is specified as the *z0* parameter in the Impact file): this will in effect isolate the lattice. This type of simulation must be carried out with a fixed timestep (which can be selected in `Snook`’s simulation options.

5.2. Results

5.2. Sputter coefficients

Sputter coefficients (γ = number of atoms sputtered per incident projectile) are typically of magnitude 0.1-10 for projectiles in the keV range. A simulation involving 300-500 runs (incident trajectories) can be completed in a few hours on a 100 MHz computer.

Some results are shown below for a 726 atom Ag(111) target bombarded with the following 0.4 keV projectiles: He, Ne and Ar (you can find the input files for the Ne/Ag(111) system in the `\examples\ne-ag111` directory of your SK installation).

After completion of the simulation, the following filter condition was applied to the output file:

`[rw > 0 ] & [vz > 0.0] & [rz > 4.5e-10] & [ke/ep > 0.2]` (5.1)

This condition only selects ejected target atoms with energies above 0.2 eV which are more than 4.5 Å above the original lattice surface boundary. The easiest way to estimate the sputter coefficient is to relate it to the size of the file (FileSize, in bytes) created by the filter operation:

$$\gamma = \text{FileSize} / (\text{RecSize}*\text{NRuns}), \tag{5.2}$$

where RecSize is the size of one binary record in `.SNK` files (40 bytes) and NRuns is the number of runs in your simulation (= 376 in this case). (The screen output of `Winnow` 2.1 displays this information automatically.)

Similar conditions as (5.1) were applied for a variety of above-surface distances (i.e. z_{min} = 4.5, 5.5, 7.0, 10.0 and 15 Å respectively, where z_{min} is the point above the surface where the sputtered atom count commences). The corresponding sputter coefficient estimates are summarised in Table 5.1.

Table 5.1. Calculated sputter coefficients for 0.4 keV projectiles normally incident on 736-atom Ag(111) target (based on 376 incident trajectories). The simulations were terminated at 400 fs, or when all particle energies fell below 1 eV. Experimental values refer to polycrystalline Ag substrates, and are taken from the compilation by Benninghoven *et al.* (Fig. 2.110).[34]

Projectile	`rz` *boundary, Å*	*Sputter coeff.,* g
He	4.5	0.84
He	5.5	0.27
He	7.0	0.17
He	10.0	0.10
He	15.0	0.05
He	Experimental	0.19
Ne	4.5	3.85
Ne	5.5	2.91
Ne	7.0	2.14
Ne	10.0	0.62
Ne	15.0	0.54
Ne	Experimental	1.6
Ar	4.5	5.66
Ar	5.5	4.37
Ar	7.0	3.19
Ar	10.0	1.93
Ar	15.0	0.95
Ar	Experimental	2.7

The data collected in table 5.1 illustrate how difficult it is to develop a practical criterion which separates the sputtered atoms from others in the plasma-like region which used to be known as the "selvedge" in the SIMS literature of the 1980s.* In these calculations, a range of 4.5 Å was used for the potential cut-off, so this is presumably a lower bound for the width of the selvedge. From comparison with the experimental values, and from visual inspection of the *z*-distribution of ejected particles, one gets the impression that the selvedge extends to around 5.5-7.0 Å. It is satisfactory to note, however, that relative yields calculated for the different projectiles are much less dependent on the choice of selvedge width (table 5.2). It is not clear from the literature how the various authors defined their sputter yields in terms of particle positions.

Sputter yields are strongly influenced if a planar surface potential (sect. 4.3.4) is employed in the simulation.

Table 5.2. Calculated ratios of sputter coefficients for 0.4 keV projectiles(He, Ne, Ar) incident on Ag(111): dependence on assumed selvedge width, z_{min} (see text for discusssion, and table 5.1 for simulation conditions and ref. to experimental data).

z_{min} (Å)	4.5	5.5	7.0	10.0	15.0	Expl.
$\gamma(Ar)/\gamma(He)$	6.8	8.1	18.4	19.3	20.0	16.9
$\gamma(Ar)/\gamma(Ne)$	1.47	1.5	1.5	1.6	1.8	1.7

5.3. Angular effects in sputtering

There are several distinct kinds of experiments which explore angular effects in sputtering processes. The simulation procedures are generally similar for each kind of experiment, although those simulations where the projectile incident geometry varies are best handled as batch jobs (see the ICISS example in the following section).

A simulation is linked to a particular experimental measurement only by the criteria used at the output-processing stage to categorise sputtered particles. For example, the output data from a given simulation could equally well be processed by `Winnow` to give a plot of either the azimuthal or the altitudinal distribution of ejected particles (or of many other kinds of distribution): the ultimate use of the output data need not be known or specified at the time that the simulation is run. This flexibility is an important advantage of the *Simulation Kit*, although it is achieved at the expense of increased complexity for the user.

`Winnow`'s predefined angular variables (`phid`, `altd`, if expressed in degrees, or `phi`, `alt`, if expressed in radians) ease the labour of extracting angular information. Suppose you want to make a plot of the azimuthal distribution of ejected particles (excluding the projectile) at an altitudinal angle of 45±3 °. Then you would follow this procedure:

1. Filter the output file (a) to exclude records for the projectile (i.e. `[rw > 0]`), (b) to select only ejected target atoms (e.g. `[rz > 4.0E-10] & [vz > 0.0]`), and (c) lastly include only those particles moving at the desired altitude (`[altd > 42.0] & [altd < 48.0]`).
2. Use the Process|Spectrum option to generate the distribution of the variable `phid` over the angular range 0.0 to 360.0.

The filtering process suggested in (1) can be achieved in a single step:

---

* I use the term *selvedge* to refer to the plasma-like surface region which assembles in classical bombardment simulations. Some writers, e.g. Robinson [7], use the term as a synonym for "surface".

```
[rw > 0] & [rz > 4.0E-10] & [vz > 0.0] &
                                          [altd > 42.0] & [altd < 48.0])
```

(5.3).

The possible range of the angular variable `altd` is -90.0 to +90.0 (degrees), but the negative values (implying motion into the bulk) are rarely needed.

Positive values of `altd` imply positive values of vz (the perpendicular velocity), which means that the term `[vz > 0.0]` in equation 5.3 can be omitted, since it is redundant.

6. Impact Collision Ion Scattering

6.1. Introduction

The repulsive interaction between an incident atomic projectile and a target atom at the surface of a lattice creates a region (known as the *shadow cone*) into which the primary projectiles cannot penetrate. Impact Collision Ion Scattering Spectrometry (ICISS) is an ion scattering technique which uses the shadow cone effect to derive structural information.[8] The shadow cone formalism has also been applied to explain projectile-incidence anisotropies in other techniques such as SIMS,[35,36] ion-induced secondary electron emission,[37] and ion-induced Auger electron emission.[38],[39]

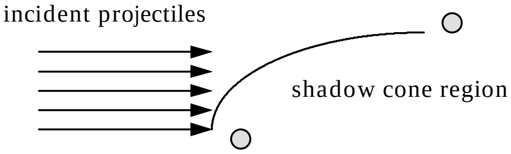

Figure 6.1. Shadow cone region formed by trajectories of projectiles scattered by repulsive potential.

The simulation of measurements made in ICISS experiments is one of the most demanding applications of the *Simulation Kit*. This is chiefly because ICISS measurements only detect backscattering of projectiles into a narrow angular range (~ 1°). For this reason, practical ICISS simulations require the use of simplified scattering models and a modest number of target atoms. Using these simplified models, a wide-range ICISS spectrum can be simulated in approximately one working day on a Pentium PC. The output from the simulation contains information about projectile scattering at all angles of emission (not just backscattering). Data for the scattering angles of interest is selected later using `Winnow`. This means that a single simulation can generate data for a variety of experimental measurements.

The simulation of ICISS data and other kinds of angular scans are good candidates for running as batch processes (using `Snook`'s batch run option), as described in section 6.2.

In ICISS experiments, a projectile is incident parallel to a row of the target lattice. Scattered projectiles are detected in the same plane, at a scattering angle of ~180°.[8] Scattering in the plane requires that the projectile's collision partners all be coplanar, so the first simplifying assumption we can make is to consider only those collisions involving projectiles incident in the plane of the atomic row. This model implicitly ignores the effects of vibrational displacements. A related simplification is to consider a target which consists only of atoms in the plane of incidence (see fig. 6.2). Finally, since ICISS is highly surface sensitive, we need only use a target consisting of a small number of atomic layers (3 or 4).

Using the first of these assumptions (and, optionally, the second and third) we can then determine the relative backscattering yield by considering only those projectile trajectories which lie in the plane of the atomic row (see fig. 6.2).

The following discussion assumes that the parameters of the projectile-target interaction potential (notably the screening length correction) are known. In practice, these will often have to be determined heuristically. Optimisation exercises of this kind should usually be carried out using a "binary search" method: for example, suppose the screening length correction is known to lie in the range 0.7 to 0.9, and the optimum value is 0.84, then the search would proceed by testing the following values: 0.8 (intermediate between 0.7-0.9), 0.75 (fails), 0.85, 0.825 … and so on.

One method for selecting the screening length correction for an ICISS simulation is to select a scattering process that involves 2 atoms (shadower and scatterer) which are both located in the surface plane, then use a binary collision program (such as `Cone`) to fit the experimental shadow cone data to a known structure (e.g. the unreconstructed clean surface). This binary collision method is based on the assumption that the projectile does not interact appreciably with other atoms (true, provided the critical angle is not too shallow).

When comparing the results of simulations with experimental data, it should be remembered that the latter involve measurements of ion yields incident in a fixed solid angle. The simulated data, however, involve scattering into a fixed angular range. The solid angle subtended by angular ranges $\Delta\varphi$ and $\Delta\phi$ is $\Delta\Omega = \Delta\varphi.\Delta\phi.\cos\varphi$.

If the simulation is treated as a 2-dimensional scattering problem (as in this chapter), no correction to the scattered intensities is required. If a full 3-dimensional calculation is undertaken, scattered intensities need to be corrected (for comparison with experimental measurements at fixed solid angle of acceptance) by dividing them by cos φ, where φ is the altitudinal emission angle.

6.2. ICISS example project

The input files you will need for the example ICISS project can be found in the `\examples\iciss` directory of your SK installation. Be sure to read the `readme.txt` files in that directory and *before* you attempt to run the simulation (which uses a batch files, `cu110.bdf`). This project simulates the scattering of 1.5 keV He from a Cu(110) surface at various altitudinal angle of incidence.

The input files for each projectile incidence angle are identical, with the exception of the Run files. The latter differ only in the specification of the projectile altitudinal angle of incidence, so it is easy to generate Run files for each geometry from a common 'template' Run file simply by editing the altitudinal angle field (using `Spider`'s Run|Open command). If you subsequently need to modify some other field in the Run file (e.g. the timestep) you may find that it is less tedious and error-prone to go back and generate the files you need from a common ancestor template, rather than edit the existing Run files.

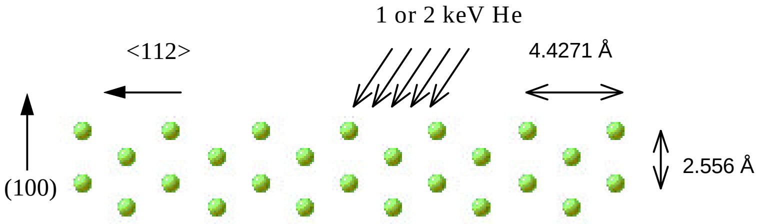

Figure 6.2. 26-atom Cu(110) lattice used for example project (projection in *xz* plane). The arrows depict the range of incident projectile trajectories in the simulation.

Fig. 6.2 illustrates the scattering geometry on a 2-dimensional target lattice. In this example we shall consider the ideal case of 180° scattering.

Fig. 6.3 shows the processed results of the ICISS simulation, namely a plot of the 180° backscattering yield as a function of the projectile incident altitudinal angle. The solid line in the figure represents the simulated data, while the dashed (blue) line represents the experimental data (arbitrarily scaled on the intensity axis) as measured by Niehus et al.[40] The backscattered projectile yield is below 200 counts (for a 0.5° counting interval per 25,000 trajectories) .

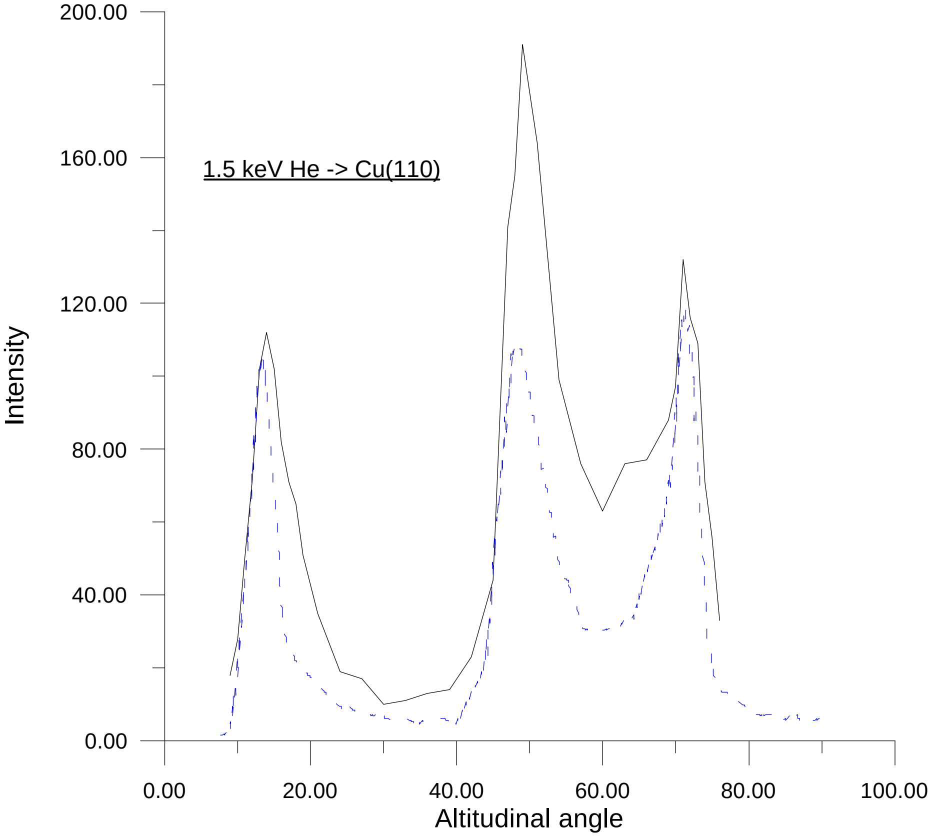

Figure 6.3. Calculated (solid line) and experimental (dashed line) direct impact backscattering intensity for 1.5 keV He projectiles incident on Cu(110) along a <112> azimuthal direction (see fig. 6.1). Yields are calculated using a ±0.5° acceptance range for the altitudinal angle (e.g., a point at φ = 60° refers to the fraction of particles backscattering between 59.5-60.5°. Each point is based on 25,000 runs (trajectories).

The data shown in Fig. 6.3 were produced by the following method:

First, in the Run file of the ICISS project, it was specified that only emitted projectile data should be written to the output file. (This restriction is not mandatory, as filtering can separate the projectile data later, but it saves unnecessary disk usage.)

Second, the screening length correction for the He-Cu ZBL potential was estimated to be 0.9 using the `Cone` utility which ships with the *Simulation Kit*. The critical angle computed by `Cone` was compared to the critical angle for in-surface scattering (12.5 °) - which is conventionally located at 85% of the experimental peak height - for various values of the screening length correction.

Third, after the simulation was completed, the resulting output file for each incident angle was filtered to select out the angular fraction of interest e.g. for 15° incidence:

`[altd > 14.5] & [altd < 15.5] & [vx > 0.0]` (6.1)

The above filter selects particles scattered in the +*x* direction at altitudinal angles (expressed in degrees) between 14.5-15.5° i.e. at an angle of 15±0.5°. (The symbol `altd` refers to the altitudinal angle *expressed in degrees*.) Since the output routine only stored projectile data in this case, there is no need to select projectile particles explicitly (i.e. by using the additional filter condition: `[rw = 0]`). This filter is appropriate for 180° backscattering events.

Fig. 6.3 illustrates the existence of 'critical angles' φ_c for 180° ('direct impact' or centre-to-centre) collisions, which are indicated by sharp onsets in scattering intensities. These appear as peaks because there is a concentration of trajectories at the shadow cone edges.

Note that if the experimental data referred to scattering at some angle less than 180°, then the filter condition should be modified accordingly. For example, for 140° degree scattering, condition 6.1 (for $\varphi = 15°$) must be modified to: `[altd > 54.5] & [altd < 55.5] & [vx > 0.0]` (Make sure you use a filter `[vx < 0.0]` if the scattered projectiles move in the negative x direction, as they would in this example for $\varphi > 50°$).

Table 3.1 compares the calculated and observed critical angles for this system. Note that the *Simulation Kit* is significantly better at locating the 70° edge than is `Cone`.

Table 3.1. Comparison of (a) observed critical angles in degrees for ICISS peaks shown in Fig.6.3 with values calculated using (b) the *Simulation Kit* and (c) `Cone` (binary collision model) respectively. Estimated uncertainties in critical angle locations are ±0.5° in columns (a-b), and ±0.1° in column (c).

Peak	(a) Observed	(b) SK	(c) `Cone`
1	12.5	13.3	12.4
2	47.2	48.0	48.2
3	70.4	70.6	71.8

6.3. Angular conventions

There is some possibility for confusion in the angular conventions used by the *Simulation Kit*. The angular variables used by `Winnow` for filtering and other options are based on a self-consistent system (see `Winnow` on-line Help for details). The user-programmed option in the Run file dialog of `Spider` also follows the same system. However, the projectile *incident* altitudinal and azimuthal angles are defined in the Run file in a manner similar to what is used by experimentalists: these angles actually define the orientation of the position vectors that join the surface 'impact point' to the starting projectile position. For this reason, the incident altitudinal angle is always entered in the Run file dialog as a positive number, although strict geometry would indicate a negative value (since $vz / \sqrt{(vx^2 + vy^2)} < 0$).[#] For 180° scattering, the projectile incident and scattering angles would both therefore be treated as positive quantities (by `Snook`/`Spider` and `Winnow` respectively).

6.4. Vibrational effects in ICISS

The ICISS example just described incorporates the effects of thermal vibrational displacements in the target lattice. These displacements increase the computational problem immensely, because of the now small probability of occurrence of a direct-impact event (which requires a scattering configuration in which both scattering atoms to lie coaxial to the projectile path).

In order to work around this problem, `Snook` offers the option of suppressing thermal vibrational displacements in the y-direction, while applying them in the normal way along the x- and z-directions. This option was used in the production of Fig. 6.3.

This workaround (the suppression of *y*-vibrations) relies on the assumption that the projectile's azimuthal incidence is parallel to the x-axis, which is achieved by leaving this parameter at its default value of zero in the Run file. Thermal displacements applied in this preferential manner maintain the coplanarity of lattice atoms in the projectile plane of incidence. To activate this option in `Snook`, it is only necessary to select the "No y-vibration" item which appears in the Simulation Options dialog box (on the Options menu).[†]

If an ICISS simulation is run without applying vibrational displacements, the critical edges will appear as sharp spikes. Some users may prefer this approach for locating the critical angles.

The simulation model shown in Fig. 6.3 fails to reproduce the relative peak heights accurately, but this is to be expected, since the model takes no account of angular variations in incident ion neutralisation efficiency. Also, the simulated width of the leftmost peak does not agree with experiment: this may be because the vibrational displacement of surface Cu atoms is wrongly estimated. Looking at Fig. 6.3, one can suggest that it is probably more useful to compare simulated and experimental data on a peak by peak basis, rather than as full angular plots.

6.5. Concluding remarks

ICCISS simulations (and others which involve many different projectile angles of incidence) require a significant post-simulation processing effort. In the case of ICISS simulations, this processing may be best achieved by automating it with a custom-written computer program, rather than relying on the general (but slow) capabilities of `Winnow`. The `examples\winnow` directory includes a simple program (`scat-cnt`) which can be used for batch extraction of the kind of data plotted in Fig. 6.3, or can be adapted for other purposes.

[#] `Spider` does not even allow you to enter a negative altitudinal angle.

[†] The "thermal vibrations" option must, of course, be enabled in the Model file used by the simulation. Otherwise the option discussed here has no effect.

7. Depth Distributions

7.1. Introduction

This chapter discusses a project which looks at the vertical displacement and spatial distribution of target particles during ion bombardment.

The input files used for this project can be found in the `\examples\mixing` directory. The project requires about 4 hours to run, and generates a very large output file (`dynvars.snk`) which is 24 MB in size. The project demonstrates the ability of the *Simulation Kit* to use a generic output file for extracting different kinds of information about a system.

Briefly, the physical system consists of a Cu(100) target which is bombarded at normal incidence by 0.5 keV Ar projectiles. The target consists of 12 atomic layers, each of which contains 225 Cu atoms (15x15x12 = 2700 atoms in total). This relatively large target was chosen in order to ensure that the collision was contained both laterally and vertically. The simulation uses 225 runs. Each trajectory is followed until either the fastest atom in the system has kinetic energy below 2 eV, or 500 fs has elapsed (in practice, the first condition is met within 300-400 fs). After the termination of each run, state information (**r**, **p**, ...etc) is recorded for each particle in the system.

The purpose of the simulation was to examine (a) the extent of vertical mixing of atoms from different layers; (b) the depth of origin of sputtered atoms; (c) the vertical distribution of implanted primary projectiles; (d) the formation of Wehner spots.

For purposes (a) and (b), the first necessary stage in data processing was to filter the output file, `dynvars.snk`, in a way that would isolate the contributions from individual layers. This can be achieved (using `Winnow`'s Filter option) for layer 1 by noting that this layer corresponds to the particles referenced by lines 1-225 of the input Target file `mixing.trg`. Thus a suitable filter expression for this layer is as follows:

`[rw > 0] & [rw <= 225]` (7.1)

Likewise, the following filter can be used to isolate the second layer:

`[rw > 225] & [rw <= 450]` (7.2)

After several applications of this procedure, one obtains a series of smaller files `layer1.snk`, `layer2.snk`, ... which contain the layer-specific data.

7.2. Sub-surface depth distribution

Fig. 7.1 shows the post-bombardment depth (z) distribution of particles plotted according to the layer of origin. The ordinate scale consists of the total counts measured for each channel (1 channel = 0.4 Å) summed over all 225 runs. The individual curves were obtained by using the Processs|Spectrum option in `Winnow`, with a “spectrum independent variable” defined as `rz*1.0E10` (i.e. the z-position expressed in Å).

The integrated area under each curve is identical, and corresponds to (225 runs x 225 particles =) 50625 counts. The most obvious feature of fig. 7.1 is the relative broadening of the curves in the outermost layers (with consequent decline in heights of the curve maxima). It is possible to discern tails on the low-z side of these curves which are indicative of recoil mixing. The extent of mixing is somewhat disguised by the contribution from particles near the edges of the target planes, which are little disturbed by the collision cascade. We shall not discuss further how to present data of this kind in order to better bring out the mixing effects, although clearly it is a delicate problem.

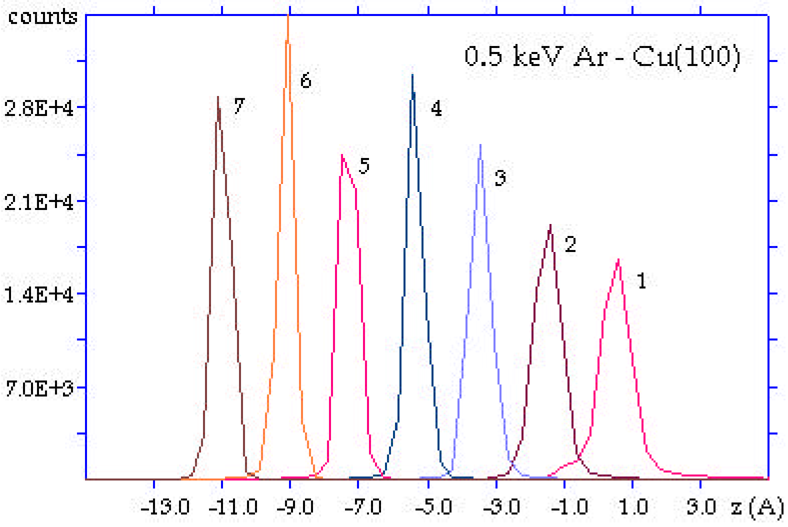

Figure 7.1. Depth distribution of target atoms in Cu(100) following 0.5 keV Ar bombardment. The figure shows the contribution (total counts from 225 runs) from particles in each of the first 7 atomic layers (225 particles per layer; 0.4 Å per channel).

7.3. Origin of sputtered atoms

Fig. 7.2 isolates the depth distribution of particles originating in layer 4 (5.4 Å) of the target. These data are the same as those shown in fig. 7.1, but now the ordinate scale has been expanded by a factor of 100. From fig. 7.2 it is evident that a negligible number of atoms from layer 4 are able to penetrate the surface (at z = 0.0 Å) and leave the target as sputtered atoms.

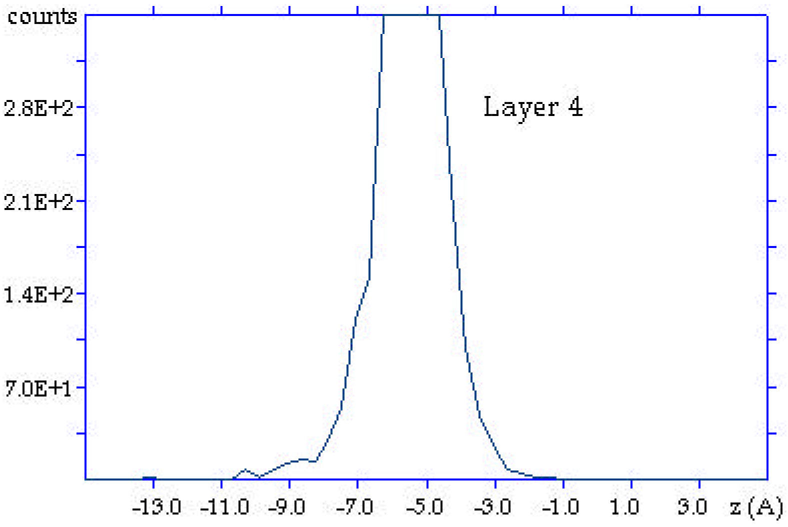

Figure 7.2. Same data (for layer 4 only) as shown in fig. 7.1, but with the ordinate scale expanded by a factor of 100. Note the total absence of layer 4 atoms found at z > -1 Å.

The depth of origin of sputtered atoms can be quantified by the following methodology. First some threshold z value must be defined which represents the beginning of the region occupied by sputtered particles, as opposed to the outermost surface atoms. As discussed in section 5.2, this definition is not a trivial task. For the purposes of discussion, we shall assume a threshold here of z = +6.0 Å. The contribution of each atomic layer to the sputtered atom yield can then be estimated by performing a further filter operation on the processed output files (`layer1.snk`, `layer2.snk` etc.) whose preparation was described above. This filter saves those particles which are located at vertical (z) positions at 6 Å or greater:

`[rz > 6.0E-10]` (7.3)

The finding of this procedure is that of a total of 1080 atoms found to be sputtered after 225 runs (a mean yield of 4.8 atoms per incident projectile[*]), 1067 of these (99%) originated from layer 1, 13 (1%) originated from layer 2, and none originated from the deeper layers (3, 4, 5...). This result emphasises the surface specificity of sputtering.

7.4. Projectile implantation profile

The projectile depth (z) distribution can be accessed from the output file (`dynvars.snk`) of the project by filtering with the expression `[rw = 0]`, followed by generation of the depth "spectrum" as described in section 7.2. Since the projectile may have left the target region (which covers a radius of ca. 17 Å from the origin), it is prudent in practice to use a filter condition like the following, which ensures that the filtered records correspond to projectiles located in the vicinity of the target:

`[rw = 0] & [sqrt(rx*rx + ry*ry) < 17.0E-10]` (7.4)

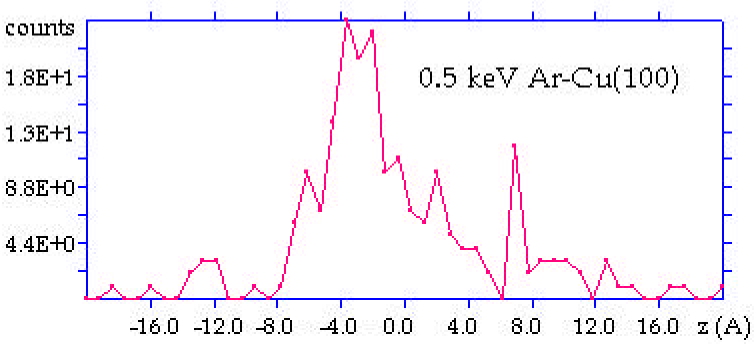

Figure 7.3. Depth (z) distribution of projectile species at termination of the simulation outlined in section 7.1 for 0.5 keV Ar-Cu(100). The figure does not include those projectiles (5.8% of total) that were laterally displaced by more than 17 Å from the lattice origin. The width of each channel is 0.8 Å.

Fig. 7.3 shows the plot which results from this procedure. The distribution is noisy because it is based on only 225 trajectories. A substantial fraction of the projectiles are reflected from the surface (ca. 20%, depending on where the surface is drawn). Of more interest to the present discussion are those projectiles which remain lodged inside the target. Fig. 7.3 shows that these are mostly stopped within 8 Å of the surface (equivalent to 4-5 Cu(100) atomic layers).

---

[*] M. Hou and W. Eckstein, Nucl. Instr. Methods B13 (1986) 507, calculated a sputter yield of 2.6 for the same system. The discrepancy with the value given here is probably related to the definition of the sputter threshold, as previously discussed in section 5.2.

7.5. Wehner spots

It has long been known that sputtered particles are preferentially ejected along directions parallel to the more closely-spaced atomic rows. This can be observed experimentally in the phenomenon of "Wehner spots."[7] Although this topic is unconnected with the subject of the chapter, it can be examined using a similar methodology, and is for that reason briefly discussed here. Wehner spots are prominent features in the images formed by sputtered particles incident upon photographic plates (or other imaging devices) facing the plane of the surface under bombardment (a set-up similar to that used for Laue or electron diffraction).

Wehner spots can be investigated through simulations by making plots of the asymptotic sputtered particle momenta in the (x,y) plane. Fig. 7.4 shows a plot of px versus py for sputtered particles in the 0.5 keV Ar-Cu(100) system. These were obtained by first filtering the output file (`dynvars.snk`) with the following expression, which isolated the sputtered target atom component:

```
[rw <> 0] & [rz > 6.0E-10]
```
(7.5)

The `SNK` file that resulted was then loaded into the trajectory plotter, from which fig. 7.4 was copied.[*] Fig. 7.4 constitutes a projection of the particle trajectories onto the (x,y) plane. The maximum spot intensities coincide with 45° emission along the <001> azimuthal directions, which correspond to the altitudinal directions parallel to <011> atomic rows.

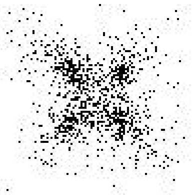

Figure 7.4. Scatter plot of sputtered particle momenta (p_x vs. p_y) for 0.5 keV Ar-Cu(100) system, showing the formation of Wehner spots. The *x* and *y* axes coincide with <011> target lattice rows. The diagonal directions lie parallel to <001> rows. The figure is based on 225 projectile trajectories.

This example will not be discussed in greater detail. An obvious first improvement of the data presented in fig. 7.4 would be to exploit symmetry and "fold over" each data point into symmetrically equivalent quadrants. This can be achieved (with some

---

[*] In the Display | Options dialog, "Momenta" was chosen. The plot shown in fig. 7.4 was created by playing through all frames, using "Points" for the symbols, and by having the "Refresh between frames" option unchecked.

difficulty) using the Format Columns option of `Winnow`,[‡] but it would better to program the operation directly in a spreadsheet or computer program.

[‡] You need to make 4 data (*`.dat`) files, using the abs() function to permute the sign of the px, py entries in turn as (+,+), (+,-), (-,+) and (-,-) . The files can be recombined using an editor. There will be considerable redundancy (duplicate entries) but this won't affect the resulting plot.

8. Thermal Vibrations

8.1. Introduction

The author became increasingly dissatisfied by the method of handling thermal vibrations in the first version of the *Simulation Kit*. The original intention was to provide an order-of-magnitude estimate of lattice atomic displacements. As the following discussion will show, the choice of atomic displacements for atomic collision simulations is a non-trivial problem. However, feedback from users indicated that a more ‘precise’ treatment was necessary, particularly for temperature variation studies. Accordingly, the basis of the computations made by `Snook` and `Spider` (in versions 1.2 or 2.1) is outlined in some detail here. The discussion in this chapter, which is certainly not the only way to approach the problem, should allow users to decide for themselves on the value of the treatments used.

Before discussing the equations, the following general points are worth drawing attention to:

1. Users can always override the mean square thermal vibrations calculated by `Spider` by simply editing the values in the .`MDL` file. (Refer to the File Formats topic in `Spider`’s online Help)
2. Most experiments based on keV ion bombardment are not particularly sensitive to thermal vibration effects.
3. Errors in the treatments of vibrational effects will only significantly influence simulations of physical processes if the root mean square (rms) vibrational amplitude *error* is significant in comparison with the scattering cross section (collision radius).
4. Most published keV bombardment simulations have not included thermal vibration effects.
5. Don’t confuse the rms vibrational amplitude $\sqrt{<r^2>}$, with the mean square thermal vibrational amplitude, $<r^2>$.
6. Don’t confuse the isotropic mean square amplitude $<r^2>$ with the unidirectional mean square amplitudes $<x^2>$, $<y^2>$, $<z^2>$.
7. Debeye temperatures are uncertain to at least ±10%, and often more.

8.2. Theory

According to the Debeye theory, the isotropic mean square thermal vibration amplitude, $<r^2>$, of atoms in a monatomic solid is given by the following expression:

$$<r^2> = 9\hbar^2 T/(Mk\theta_D^2).\ [\ \phi(\theta_D/T) + \theta_D/4T] \qquad (8.1)$$

where θ_D is the Debeye temperature, k is Boltzmann’s constant, T is the absolute temperature and M is the mass of atoms in the solid. $\phi(x)$ is the following function, where $x = \theta_D/T$:

$$\phi(x) = (1/x)\cdot\int_0^x \frac{y\,dy}{(e^y - 1)} \qquad (8.2)$$

Equation 18.1 refers to the mean square amplitude of an isotropic oscillator. This is the quantity which `Snook` reads from the .MDL file created by `Spider`. For vibrations in a specific direction (x, y or z), the right hand side of equation 18.1 must be divided by 3:

$$\langle x^2\rangle = \langle y^2\rangle = \langle z^2\rangle = \langle r^2\rangle / 3 = 3\hbar^2 T / (Mk\,\theta_D^2)\ .\ [\ \phi(\theta_D / T) + \theta_D / 4T] \tag{8.3}$$

These unidirectional amplitudes are the quantities which are calculated and used by `Snook`.

If we substitute for the physical constants, and interpret M as the mass in atomic mass units (amu), then with the substitution ($x = \theta_D / T$) the expression (18.3) simplifies to:

$$\langle x^2\rangle = 145.5 / M\theta_D\ .\ [\ \phi(x) / x + 1/4]\quad Å^2 \tag{8.4}$$

[$\hbar$ = 1.05459E-34 Js; k = 1.38066E-23 JK^{-1}; 1 amu = $(1000N_A)^{-1}$ = 1.6606E-27 kg]

8.3. Comments on the Literature

There is some confusion in the equations published in the literature, in that equation 18.3 is sometimes associated with the isotropic vibrations (more properly described by equation 18.1). This presumably occurs because of typographical errors. Despite this, and other discrepancies, most authors seem to broadly agree on the constant pre-factor (5.5) in equation 18.4 (although physicists tend to calculate the pre-factor using nuclear rather than atomic masses, giving a value of 6.0). The reader is urged to be cautious when reading the literature on this subject (including this article!).

8.4. Implementation in `Spider` and `Snook`

`Spider` (versions 1.2 and 2.1) uses equation 18.4 to calculate $\langle x^2\rangle$ for bulk and surface atoms, based on the values for T and θ_D specified by the user (for the Model file), and the approximations to $\phi(x)$ discussed below. These unidirectional $\langle x^2\rangle$ values are read by `Snook` from the Model file. The displacements added to each lattice atom *x*, *y* and *z* coordinate are calculated (using the Box-Muller method) to be consistent with a random distribution having Gaussian deviates characterised by a variance $\langle x^2\rangle$. Bulk, surface parallel and surface perpendicular displacements respectively are drawn from distributions with different standard deviations. However, a requirement imposed on the displacements calculated by `Snook` is that they shall not exceed 2.5 standard deviations. For purposes of the vibrational correction, a lattice atom is classified as a ‘surface atom’ if its z-coordinate, z[n], places it above the anchor atom at `z`[1], or no more than 1 Å below it: [1]

```
if z[n] >= z[1] - 1.0E-10 then {the atom is in a surface site }
else { atom is in a bulk site }
```

The method for calculation of the Debeye function $\phi(x)$ in equation 18.4 is now explained.

$\phi(x)$ can be calculated by numeric integration of equation 18.2. Eckstein provides the following approximation:[3]

---

[1] You should contact the author if this treatment does not meet your requirements.

$$\phi(x) = (1/x)\cdot\int_0^x \frac{ydy}{(e^y-1)} = 1 - x/4 + \frac{x^2}{36} - \frac{x^4}{3600} + \ldots \qquad (8.5)$$

This approximation is used by Snook for x in the range 0 to 3.0, i.e. in the range T = $0.33\theta_D$ to "infinity". Recall that $x = \theta_D/T$. For "infinite" T, $\phi(x) = 1.0$; for T = $0.33\theta_D$, $\phi(x) = 0.48$.

As $T \to 0$, so too $\phi(x) \to 0$. For values of $x > 3.0$ the following (low-temperature) approximation is used:[3]

$$\phi(x) = [\pi^2/6 - (x+1)\exp(-x) - (x/2 + 1/4)\exp(-2x) - \ldots]/x \qquad (8.6)$$

For $x >> 1$, $\phi(x)/x$ becomes small in comparison to the ¼ term in equation 18.4, and the lattice vibrations approach their zero-point levels.

It is worth pointing out that these thermal displacements are computed without reference to the potential function (Morse) used for the target-target interactions in your lattice. This means that the (thermo-)dynamical temperatures associated with your lattice (which depend on the potential parameters) will not correspond to the temperature which you specified in the model file. The true lattice temperature is related to the mean *difference* in potential energies, $<V_1 - V_0>$, of the lattice with (V_1), and without (V_0), thermal displacement effects applied. For a system of oscillators interacting via quadratic (harmonic) terms, theory suggests a value of $<V_1 - V_0> = 3/2kT$ (which we shall use as an approximation for the Morse potential too; this result comes from either the Equipartition principle,[41] or the Virial theorem,[9] depending on your point of view). The discrepancy between theory and the values returned by Snook depend on the potential function used by Snook. For example, the default Morse potential for Cu gives a value of $<V_1 - V_0>$ = 0.076 eV (300 K), compared with 3/2kT = 0.04 eV.

Further discussion of these matters would take us too far afield, but the conclusion must be that the choice of an appropriate thermal displacement depends to some extent on the purpose of the simulation. For a study of ion scattering or channelling, the primary focus of interest is the displacement itself. However, for study of lattice dynamics (melting, film growth, diffusion etc.), the potential energy associated with (or implied by) the displacements should be the primary consideration. Sputtering - one of the main applications of such simulations - unfortunately falls in the area between these extreme cases.

8.5 Lattice atom velocities

The simulation options dialog box in Snook allows the user to initialise the velocities of the target lattice atoms. The idea is to set-up the target lattice as a dynamical system of a specific temperature (see, however, the preceding section for qualifications to this remark). The velocities are randomly applied, according to a Maxwellian distribution.[‡] Note that this velocity initialisation is based on a classical theory, in contrast to the quantum mechanical foundation of the Debeye theory on which the thermal

---

[‡] No attempt is made to ensure that $<v_x>$, $<v_y>$ and $<v_z>$ are precisely zero, although in practice this very nearly happens in large lattices, as expected.

displacements are estimated.

According to the Equipartition Principle (from statistical mechanics), the mean translational energy <KE> of an atom in the classical limit is 3/2 kT (k is the Bolzmann constant). This can be broken down into contributions of ½kT for each direction of motion. The velocity distribution is deduced from the following assumptions:

(a) atomic energies are distributed according to a Boltzmann distribution: i.e. $P(E) = \exp-((KE + V)/kT) = \exp(-KE/kT).\exp(-V/kT)$;

(b) atomic positions and momenta are distributed independently: i.e. $P(E) = P(V).P(KE)$.

9. Inelastic Scattering

9.1. Introduction

Until this point the discussion has considered only the effects of elastic scattering. Energetic particles can lose energy through excitation of excitation of valence or core electrons, or lattice vibrations. These energy losses are termed inelastic losses. The term *electronic stopping* is also used to refer to electronic excitations, which are the predominant loss mechanism at keV particle energies and above. The reader should be aware at the outset that the main pitfall in the literature of electronic stopping is the lack of any standard system of units. The equations in this chapter should be used with SI units.

Version 2.2 of the *Simulation Kit* permits the modelling of inelastic effects via a combination of three distinct electronic stopping models.[4] These models are: (a) the Lindhard-Schiott-Scharrf (LSS) model; (b) the Oen-Robinson (OR) model; (c) the Shapiro-Tombrello (ST) model.[32,33] These models can be included singly, not at all, or in any weighted combination according to the preference of the user. The simulation of inelastic effects at keV energies usually has to be done on an *ad hoc* basis because of the general theoretical uncertainty surrounding the mechanisms of inelastic loss. The parameters for each of the models (a)-(c) are specified using `Spider`; the actual model(s) to be used are selected in `Snook's` Options box at run-time.

The LSS model requires the specification of a set of parameters for every kind of atom (i.e. atomic number, Z1) for which inelastic losses are to be tracked. For instance, 2 sets of parameters are required for the Ar → Cu(100) system (for Ar and Cu particles respectively). The OR and ST models require a set of parameters for each pair of collision partners. For example, 3 sets of parameters are required for the Ar → Cu(100) system, representing the pairs Ar-Ar, Ar-Cu and Cu-Cu respectively. For all models, incomplete sets of parameters are allowed, but this will lead to neglect of the corresponding energy loss channel (for example, if you forget to include parameters for the Cu-Cu interaction). The `Spider` online Help topics explain how you should enter the parameters for these various inelastic loss models.

In implementing these models, the programmer (i.e. the author) is forced to make decisions with which the user of a program may not necessarily agree. For example, in `Snook`, any inelastic loss corrections for a given timestep are applied *after* the update of the elastic corrections (positions and velocities) for that timestep. The distinction may seem trivial, but it does have observable consequences.

9.2. Lindhard-Schiott-Scharrf (LSS) model

The LSS model asserts that the electronic energy loss (dE) associated with the movement dx of a particle (atomic number Z_1) in a medium (atomic number Z_2) is proportional to the particle velocity (v):

$$-\frac{dE}{dx} = 8\pi N\left(\frac{e^2}{4\pi\varepsilon_0}\right).a_B \frac{Z_1^{7/6} Z_2}{(Z_1^{2/3} + Z_2^{2/3})^{3/2}} v / v_B \qquad (9.1),$$

where N is the atomic density of the medium (atoms m^{-3}), a_B is the Bohr radius, and v_B is the Bohr velocity (= $c/137$). Equation 9.1 simplifies to:

$$-\frac{dE}{dx} = K(LSS).v \qquad (9.2),$$

where K(LSS) is a constant. The LSS model thus views the target as a viscous medium, and describes a continuous electronic energy loss process that arises from passage through this medium. Spider allows the definition of K(LSS) given in equations 9.1 and 9.2. to be scaled by an arbitrary factor. dE can be calculated at each timestep, once an atom's displacement, dx, is computed for that timestep.

For a mixed target material (e.g., a Cu-Ni alloy), the correct choice of the Z_2 parameter in equation 9.1 requires some careful thought, which will be guided by the user's physical intuition about the correct 'effective' atomic number of the target.

It should be noted that the application of inelastic corrections according to the LSS model does not conserve linear momentum. (This remark does not apply to the OR and ST models.)

9.3. Oen-Robinson (OR) Model

The OR model estimates the energy loss (ΔE) arising from a single isolated binary atomic collision in which the distance of closest approach (apsidal distance) is R_0:

$$-\Delta E = 8\pi(\frac{e^2}{4\pi\varepsilon_0}).a_B \frac{Z_1^{7/6} Z_2}{(Z_1^{2/3}+Z_2^{2/3})^{3/2}} v/v_B \frac{(c/a)^2}{2\pi} e^{-(c/a).R_0} \qquad (9.3)$$

Here c is a constant term (normally c = 0.3, but this can be specified arbitrarily in `Spider`), and a is the uncorrected Moliere-Lindhard screening length. Z_1 is the atomic number of the 'projectile' species, while Z_2 is the atomic number of the 'target' atom species. (The roles of the two atoms involved in the collision can be symmetrized in `Spider`, if the user so desires.)

As implemented by Snook, the term v in equation 9.3 represents the relative velocity ($|\mathbf{v}_1 - \mathbf{v}_2|$) of the collision partners at 'infinite separation', which is computed as follows:

$$v = \sqrt{\frac{1}{2}\mu.v(t)^2 + V(r(t))} \qquad (9.4),$$

where $v(t)$ and $V(r(t))$ respectively represent the magnitude of the relative velocity at some time t, and the potential energy at the same time (the apsidal point is used by `Snook`); μ is the reduced mass of the system.

The connection between equations 9.1 and 9.3 is brought out if the latter is written in the form:

$$-\Delta E = 8\pi(\frac{e^2}{4\pi\varepsilon_0}).a_B \frac{Z_1^{7/6} Z_2}{(Z_1^{2/3} + Z_2^{2/3})^{3/2}}.F_{OR}.v / v_B \qquad (9.5),$$

$$-\Delta E = K(OR).F_{OR}.v \qquad (9.6),$$

where F_{OR} is the 'Oen-Robinson factor,' and $K(OR)$ is a constant for a given collision pair. Note the absence of any atomic density (N) term in equations 9.3 and 9.5. *K(OR)* has very different physical dimensions from *K(LSS)* defined in the preceding section. A scale factor may optionally be specified in `Spider` to adjust the magnitude of the energy loss described by equations 9.3, 9.5 and 9.6.

In contrast to the LSS model, the OR model has the character of a discrete energy loss, because it associates a single loss event with each close encounter. One problem with implementing the OR model is in finding a technique to deal with collision events that cannot be approximated as binary encounters. `Snook` circumvents this problem, perhaps inelegantly, by requiring the user to impose (heuristically) an upper limit R_{MAX} on R_0 (the distance of closest approach). Thus, the OR energy loss is not computed unless the colliding atoms approach within a distance R_{MAX}. Typically, R_{MAX} would be on the order of 1-2 Å.

The way in which the computed OR energy loss (ΔE) is applied within the simulation routine is identical to the method used for the ST model described in the following section (qv.).

9.4. Shapiro-Tombrello (ST) model

The ST model attempts to incorporate collision-induced core electron promotion effects into a classical dynamical scattering model. The main drawback of the model is the uncertainty surrounding the correct choice of parameters (p, R_C, ΔE: see below). A brief summary of the model as implemented in `Snook` will now be given. For a deeper review of the physics involved, the reader is referred to the original literature.[32, 33, 42]

The idea underlying the ST model is that an inelastic (inner-shell electron promotion) transition can occur if a colliding pair of atoms approaches closer than some critical distance (R_C). This energy loss may involve up to N_{MAX} electrons from the inner shell. For each electron promoted, an amount of inelastic energy ΔE is lost (the maximum loss possible is thus $\Delta E * N_{MAX}$). The number of electrons considered for promotion (N) depends on the relative *radial* kinetic energy ($K_R = ½\mu v_R^2$) available to the collision pair at the moment when R_C is passed; i.e., N must be consistent with the condition: $K_R \geq N\Delta E$. Finally, for each of the N electrons, a probability factor (p) is compared with a random number to determine whether or not that electron is actually promoted (for instance, if $p = 0.5$, typically only ~N / 2 electrons will be promoted).[*]

From the foregoing, we see that the total energy loss (ΔE_T) computed for a particular collision configuration satisfies the conditions:

$$\Delta E_T \leq N\Delta E;\ \ \Delta E_T \subset \{\Delta E, 2\Delta E \ldots N\Delta E\} \qquad (9.7),$$

---

[*] The author is indebted to Dr Shapiro for this suggestion.

(the equality applies if $p = 1$, in which case $\Delta E_T = N\Delta E$), while the average energy loss over many such collision configurations is:

$$<\Delta E_T> = N*p*\Delta E \quad (9.8).$$

The inelastic energy loss corrections for both the ST and OR models are applied at the apsis of the collision.[#] At the apsis, the radial kinetic energy (K_R) is zero. The energy loss correction (ΔE_T) is applied by reducing the potential energy of the interacting atoms: this entails translating them instantaneously along the line joining their centres by a distance Δr, which changes the potential energy by an amount ΔV, such that $\Delta V = \Delta E_T$.

The translation distance Δr was originally estimated by ST via the relation: $\Delta r = \Delta V / F(r)$, where $F(r)$ is the force at the apsidal point.[32] This equation would be exact if the potential declined linearly with separation (r). Snook uses a similar procedure, but applies it twice (both at the apsidal point, and at the first estimated displacement). In addition, the first application of the formula uses a heuristic correction factor of 1.2 to compensate for the rapid decline the force as r is increased. The procedure used by Snook typically computes the correct displacement Δr to within $\sim 10^{-4}$ Å or better, but such is the nature of the potential that errors of this magnitude do lead to errors at the ~1 eV level in the energy book-keeping routines. For this reason, it is important that you initially test the parameters of your simulation (in particular, the timestep) with the ST and OR loss effects disabled. This will give a true estimate of the integration error. Subsequent errors in energy conservation can then be attributed to inelastic loss book-keeping errors. The ST and OR energy loss algorithms conserve linear momentum, but not angular momentum.[2]

[#] In practice, at the first timestep after the apsis. It could be argued that the ST correction should be applied at the moment that R_C is crossed, rather than at the apsis. However, in practice this procedure would make very little difference to the collision dynamics (and no difference at all for direct impact collisions).

10. Advanced Topic: Flags

10.1. Introduction

Version 2.2 of `Snook` uses an array of bytes ('options flags') to store information relating to individual particles (1 byte per particle). Each byte consists of eight bitmapped parameters. That is, the specific bits of the byte each contain information about different options in use for the corresponding particle. The individual bits, more usually known as (options) flags (and designated by names of the form `ofXXXX`), can either be set (equal to 1) or cleared (equal to 0). The flags currently defined in version 2.2 of the *Simulation Kit*, and their values are listed in Table 10.1.

Table 10.1. Options flags (`ofXXXX`) and their values.

Flag name	Value	Binary representation	Applies to
`ofEmitted`	1	00000001	All particles
`ofNoForce`	2	00000010	Target particles only
`ofRepulsive`	4	00000100	Target particles only
`ofNotContained`	8	00001000	All particles

The options bytes are read in from the Target and Projectile files of a simulation project (as the last numeric column). The default value for all flags is zero. **Most users will have no reason to change this default behaviour, and can ignore this chapter.**

The effects of the various flags are explained in the following sections. Note from Table 10.1, that not all flags affect the projectile.

Should you wish to incorporate one or more of the associated options into your simulation, you simply need to set the corresponding bits of the options byte in the Target file and / or Projectile file of your simulation project.[*] To set bits, you must total up the respective numbers in the Value column of Table 10.1. Thus an options byte of 3 means that the `ofEmitted` and `ofNoForce` bytes have both been set (1+2 = 3).

10.2. `ofEmitted`

This flag is set by `Snook` after (a) a projectile or target particle has been ejected from the surface region of the target, and (b) a surface binding energy correction has been applied. The effect of setting this flag manually is to override (ignore) the surface binding energy correction. However, if the surface binding energy is zero (0.0), the flag has no effect.

10.3. `ofNoForce`

This flag suppresses the interaction between 2 target atoms in the Morse potential region *only*. That is, if the flag is set for *either* of the colliding atoms, the interaction between them will be ignored in this region. This flag only has an effect if the target is *complex* (consists of more than one kind of atom). The flag has no effect in the spline or screened Coulombic region of a composite potential (it would not make sense

---

[*] The options byte can be specified at the time that the Target / Projectile file was created (via the 'Flags' input box), or by manual editing in a text editor.

physically to neglect these stronger interactions). The `ofNoForce` flag may be usefully set for an atom that interacts weakly or repulsively with its neighbours, such that the Morse potential would be a poor representation of the potential (for example, Xe atoms in a system consisting of a Xe monolayer adsorbed on a Cu surface).

10.4. `ofRepulsive`

This flag affects the interaction between 2 target atoms in the Morse potential region only. It replaces these interactions with a screened Coulombic interaction, but only when *both* interacting atoms have the `ofRepulsive` flag set. This flag takes precedence over the `ofNoForce` flag. This flag only has an effect if the target is complex (consists of more than one kind of atom). Consider the following flag settings for the collision system $\mathrm{Ar} \rightarrow \mathrm{Cl}/\mathrm{Cu}(100)$:

Atom	`ofNoForce`	`ofRepulsive`
Cu	Clear	Clear
Cl	Set	Set

Based on these flags, the atoms in the system will interact at short range (i.e. the Morse region defined in the Model file) according to the following scheme:

Interaction pair	Interaction potential
Cu-Cu	Morse
Cl-Cu	None
Cl-Cl	Screened Coulombic

The benefit of this approach is that it results in a stable target configuration (apart from the Cl-Cl interactions, which are presumably weak because of the large Cl-Cl distance). The other approach, of treating the Cu-Cl and Cl-Cl interactions with the same Morse potential as the Cu-Cu interaction has obvious difficulties, although it must be admitted that these are irrelevant for certain kinds of simulations (e.g. projectile ranges, ISS and other fast processes).

10.5. `ofNotContained`

This flag is set by `Snook` after (a) a projectile or target particle has been ejected from any of the 4 sides of the target lattice, and (b) a bulk binding energy correction has been applied. The effect of setting this flag manually is to override (ignore) the bulk binding energy correction. However, if the latter is zero (0.0), the flag has no effect.

INDEX

References

---

[1] T.G. Wynn, *The Evolution of Tools and Symbolic Behaviour*, in *Handbook of Human Symbolic Evolution*, A. Lock and C.R. Peters (eds.), Oxford University Press, 1996.

[2] R. Smith, M. Jakas. D. Ashworth, B. Oven, M. Bowyer, I. Chakarov and R. Webb, '*Atomic and Ion Collisions in Solids and at Surfaces*,' Cambridge University Press, 1997.

[3] Wolfgang Eckstein, '*Computer Simulation of Ion-Solid Interactions*,' Springer-Verlag, Berlin, 1991.

[4] E.S. Mashkova and V.A. Molchanov, '*Medium-Energy Ion Reflection from Solids*,' North-Holland, Amsterdam, 1985.

[5] D.E. Harrison Jr., '*Sputtering Models - a Synoptic Review*,' Radiation Effects, vol. 70 (1983) 1-64.

[6] R. Smith and D.E. Harrison Jr., '*Algorithms for Molecular Dynamics Simulations of keV Particle Bombardment*,' Computers in Physics, vol. 3 (1989) 68-73.

[7] M.T. Robinson, *Theoretical Aspects of Monocrystal Sputtering, in Sputtering by Particle Bombardment I* (pp. 73-144) , R. Behrisch ed. (Springer-Verlag, Berlin, 1981).

[8] H. Niehus, W. Heiland and E. Taglauer, '*Low-energy Ion Scattering at Surfaces*,' Surface Science Reports vol. 17 (1993) 213-303.

[9] Herbert Goldstein, *Classical Mechanics*, 2nd ed. (Addison-Wesley, Reading MA, 1981).

[10] L.D. Landau and E.M. Lifshitz, *Course of Theoretical Physics, Volume 1: Mechanics*, 3rd ed. (Pergamon, Oxford, 1976).

[11] H.S. Tan and M.A. Karolewski, Nuclear Instr. Meth. B73 (1993) 163.

[12] M. Gryzinski, Phys. Rev. 138, (1965) A305.

[13] M.A.D. Fluendy and K.P. Lawley, *Chemical Applications of Molecular Beam Scattering* (Chapman and Hall, London, 1973).

[14] W.J. Duffin, *Electricity and Magnetism*, 4th ed. (McGraw-Hill, New York, 1990).

[15] M. T. Robinson and I.M. Torrens, Phys. Rev. B9 (1974) 5008.

[16] J.F. Ziegler, J.P. Biersack and U. Littmark, *The Stopping and Range of Ions in Matter*, vol. 1 (Pergamon, New York, 1985).

[17] R.S. Daley, D. Farelly and R.S. Williams, Surface Sci., 234 (1990) 355; S.J. Anz and R.S. Williams, Surface Sci., 372 (1997) L323.

[18] W.H. Press, B.P. Flannery, S.A. Teukolsky and W.T. Vetterling, *Numerical Recipes in Pascal* (editions also in Fortran or C) (Cambridge University Press, Cambridge, 1989).

[19] Derek Wood, *Data Structures, Algorithms and Performance* (Addison-Wesley, New York, 1993).

[20] Robert Sedgewick, *Algorithms* (2nd Ed.), (Addison-Wesley, Reading, MA, 1988), chapter 26 ("Range Searching").

[21] K. Gärtner et al., Nuclear Instr. Meth. B 102 (1995) 183.

[22] J.W. Rabalais, H. Bu and C.D. Roux, Phys. Rev. Lett. 69 (1992) 1391.

[23] K. Broomfield, R.A. Stansfield and D.C. Clary, Surface Sci. 202 (1988) 320.

[24] K. Nordlund, N. Runeberg and D. Sundholm, Nuclear Instr. Meth. 132 (1997) 45.

[25] R. Feynman, Lectures on Physics, Vol. II (sect. 39-5), Addison-Wesley, Reading, MA, 1964.

[26] G.J. Ackland, G. Tichy, V. Vitek and M.W. Finnis, Phil. Mag. A 56 (1987) 735.

[27] N.W. Ashcroft and N.D. Mermin, *Solid State Physics*, (Saunders College, Philadelphia, 1976).

[28] R.A. Gibbs, S.P. Holland, K.E. Foley, B.J. Garrison and N. Winograd, Phys. Rev. B24 (1981) 6178.

[29] Yoshitaka Kimura, Tahir Cagin, William A. Goddard III, "Properties of fcc Metals from Many Body Potentials," to be submitted to Phys. Rev. B.

[30] D.E. Harrison, Crit. Rev. Solid State Mater. Sci. 14 S1 (1988) 1.

[31] R. Smith, D.E. Harrison and B.J. Garrison, Phys. Rev. B40 (1989) 73.

[32] M.H. Shapiro and T.A. Tombrello, Nuclear Instr. Meth. B 94 (1994) 186.

[33] M.H. Shapiro and T.A. Tombrello, Nuclear Instr. Meth. B 102 (1995) 277.

[34] A. Benninghoven, F.G. Ruedenauer and H.W. Werner (eds.), *Secondary Ion Mass Spectrometry* (Wiley, Chichester, 1987).

[35] M.J. Witcomb, Radiat. Eff. 27 (1976) 223.

---

[36] C.C. Chang, Surf. Interface Anal. 15 (1990) 79.

[37] B.A. Brusilovosky, Vacuum 35 (1985) 595.

[38] L. Wong, P.F.A. Alkemade, W.N. Lennard and I.V. Mitchell, Nucl. Instr. Meth. B45 (1990) 637.

[39] R. Pfandzelter and J.W. Lee, Nucl. Instr. Meth. B45 (1990) 641.

[40] R. Spitzl, H. Niehus and G. Comsa, Surf. Sci. Lett. 250 (1991) L355.

[41] C. Kittel, *Elementary Statistical Physics*, (John Wiley, New York, 1958).

[42] T.A. Tombrello, Nuclear Instr. Meth. B 102 (1995) 312.