SOFTWARE ACCELERATION TECHNIQUES FOR THE SIMULATION OF SEM IMAGES

Adam Seeger, Charalampos Fretzagias, Russell Taylor
Department of Computer Science
University of North Carolina at Chapel Hill

Summary: An SEM (Scanning Electron Microscope) simulator was developed based on the models used in the MONSEL software\textsuperscript{21,23}. This simulator extends earlier work by introducing an object-oriented framework and adding optimization methods based on precomputation of electron trajectories. Several optimizations enable speedup by a factor of 5-100 on a single processor over unoptimized simulations without introducing additional approximations. The speedup for a particular surface depends on the self-similarity of the surface at the scale of the electron penetration depth. We further accelerate by parallelizing the calculations for a total speedup of about 100-2000 on 30 processors. The goal of this work was to create a system capable of simulating a quantitatively accurate SEM image of a relatively unconstrained surface. Results of this work include simulation software, optimization algorithms, performance measurements with various optimizations, and examples of simulated images.

Key words: Monte Carlo, simulation, scanning electron microscope, secondary electrons

PACS codes:
02.70Uu Applications of Monte Carlo methods
05.10.Ln Monte Carlo methods
07.05.Tp Computer modeling and simulation
34.80.-i Electron scattering
68.37.Hk Scanning electron microscopy (SEM)
79.20.Hx Secondary electron emission

SEM signal formation and modeling considerations

An SEM acquires images of a sample by bombarding it with a sharply focused beam of high-energy electrons and measuring the electrons that come back out after some series of interactions with the sample material. As the beam is scanned over the surface (in the same way the electron beam in a CRT is scanned across a screen), the SEM builds up an image from the measured signal. Because the distance from the sample surface to the pivot point for the beam (working distance) is typically on the order of millimeters while the region scanned is typically on the order of micrometers, the change in direction of the beam is very small over a scan region (<0.05°). Therefore, it is reasonable to use a parallel projection in modeling the SEM.

The interactions between an electron beam and a sample are very complicated and result in signals that simultaneously measure many different material and geometric aspects of a sample. When an electron enters the sample, it interacts with

Address for reprints:
Adam Seeger
Department of Computer Science
University of North Carolina, Chapel Hill
Chapel Hill, NC 27599-3175
e-mail: seeger@cs.unc.edu
some combination of electrons bound to atoms, free (conduction band) electrons, and atomic nuclei. A scattering electron will either escape to hit either the detector or the inside of the specimen chamber or will eventually run out of energy and be captured in the sample. Electrons emitted from the sample at high energy (>50eV), known as back-scattered electrons (BSE), are the most easily understood and can be modeled using relatively simple approximations derived from classical views of the atom. Low energy electrons (<50eV) emitted by the sample are commonly referred to as secondary electrons (SE) because they consist primarily of electrons that were part of the sample and have been kicked out by the incoming beam electrons. Accurate models for secondary electron production and transport are more difficult to construct because there are numerous mechanisms by which they can be produced and the physics for many materials is poorly understood. Experimental results on electron scattering at low energies are sparse and what results are available often contain a great deal of error. This error is evident in the variation of measured values for the same theoretical values reported by different researchers. Modeling secondary electrons is also complicated because their trajectories are strongly influenced by charging, contamination, and electric fields in the specimen chamber. While it is usually safe to approximate the trajectories of BSEs in a vacuum as straight lines, SE follow significantly curved paths as a result of electric fields resulting from sample charging or bias voltage applied to the detector. Some researchers attempted to model electric fields in a specimen chamber but such an approach is complicated by the dynamic nature of the charging phenomenon. Another approach is to make the approximation that escaping BSEs follow straight lines while escaping SEs always follow curved paths that take them to the detector (this detector is held at a more positive potential than the sample in order to attract electrons). We use this last approach because it is the one used by MONSEL.

Because of the complexity of beam-specimen interactions, it is easy to misinterpret what one is observing in an SEM. For the same reason it can be difficult to see from images alone what imaging conditions are ideal for studying a particular aspect of a sample. Therefore, two main reasons for simulation are to help interpret the meaning of SEM signals and to determine ideal imaging conditions.

Various imaging parameters such as magnification, beam width, beam current, detector geometry, and accelerating voltage can have significant effects on what is measured but these effects may not be intuitive. Simulation of various imaging conditions can help with determining how to set imaging parameters for a given sample. For example, given a sample that consists of a substrate of one material covered by a thin film, it could be useful to know what part of the signal comes from electrons scattering only in the thin film and what part comes from electrons that have scattered in the substrate. Jiang et al included detector characteristics in their simulation and performed virtual experiments with different imaging conditions to help determine the instrument settings that would maximize material contrast.

Through simulation, it is possible to test various hypotheses about a model of the sample by comparing simulated images with the real image. Gauvin et al used simulation techniques to help with determining the relationship between SEM images and the depth and radius of spherical inclusions in the sample. An important example of such an application is in the area of nanometer-scale dimensional metrology. It is useful to have a precise measurement of sample shape when one is trying to manufacture nanometer scale devices to match some specification or when one wishes to manipulate a sample. Knowing the shape of a surface precisely allows a precise plan to be formulated for modification of that surface and can provide feedback as to how the manufacturing process should be modified to improve the resulting surface.

Applications of SEM image simulation

Sample representation
Most software currently in use for simulating SEM images is limited to simple sample shapes described by only a handful of parameters such as thickness of various layers, slope, sphere radius and pit depth. One of the more complex examples of such a model describes the profile of an etched line or a set of closely spaced lines with parameters such as wall slope, radii of curvature for various corners, line height and line width (Davidson, 1999). While most software assumes very specific forms for the sample, a few researchers have considered more general representations. Aristov (1991) proposed a height field representation $z = f(x,y)$ but his signal model assumed that the variations in $z$ were small relative to the range of electrons scattering in the sample. One of the most general models was created by Russ et al (1990) and allowed a sample to be specified in terms of a two-dimensional cross section composed of multiple polygonal surfaces with multiple regions containing different materials. Takeuchi et al (1989) simulated a line-scan of a surface that was measured independently from a cross-section SEM image and compared this with actual line scans.

We represent the sample in terms of a height field, a two-dimensional array of height values. The array of points is tessellated with triangles to create a surface. We assume a single material filling the volume beneath the surface. As shown by Musgrave (1990), any test for ray intersection with a height field can be done in $O(\sqrt{N})$ time where $N$ is the number of height values for the surface. This efficient algorithm was implemented as part of this work. The operation of the intersection test can be seen intuitively in Figure 1.

**Figure 1:** Top-down view of height field and ray being tested. Only the shaded triangles can possibly be intersected by the ray.

We have focused our work on implementing intersection routines for the height field model because that particular representation is quite simple and general and is sufficient for our application. However, our software is designed so that other types of sample models can be defined by providing a subroutine that computes the nearest surface intersection and materials at that intersection given a ray representing part of an electron trajectory. To demonstrate this we also implemented a simulator that works with arbitrary numbers of closed surfaces of varying materials. For an arbitrary surface model additional information about surface connectedness must be provided to take advantage of one of our acceleration techniques.

**Simulation methods**

Earlier work related to the acceleration of electron scattering simulation took a variety of approaches. The simplest models of SEM signals are slope-based descriptions as used by Jones and Taylor (1994) and Reimer et al (1987) for surface reconstruction. Slope-based models fail when the surface cannot be considered to have a single slope at the scale of the beam-sample interaction volume. A more complicated model that can account at least qualitatively for much of the topography-dependence of SEM signals was invented by Archard (1961). The Archard model
is based on a spherical region embedded in the sample and representative of the interaction volume. The back-scattered electron signal is approximately proportional to the volume of this spherical region that lies outside the sample as shown in Figure 2. The depth of the center of the sphere in Archard’s original model was taken as the depth at which $\cos(\theta) = 1/e$, where $\theta$ is the cumulative scattering angle over multiple collisions and $e$ is the base of the natural logarithm. The radius of the sphere was assumed to be such that the depth of the center plus the radius was equal to the total electron range (a constant given the material and initial electron energy)." 

Scattering models based on Monte-Carlo simulation are more accurate than approximations such as Archard’s model. The Monte Carlo method consists of simulating many scattering electrons and computing various statistics such as how many electrons escape the sample. For each electron, a trajectory is computed. At each step of a trajectory, a distance to the next scattering event and the outcome of that event (scattering angle or secondary electron generation) is computed by randomly sampling from distributions described by a statistical model (Figure 3). 

Figure 2: Archard’s model. Signal is proportional to the part of the sphere that lies outside the surface.

Figure 3: Monte Carlo simulation. Filled circles represent inelastic scattering events; empty circles represent elastic scattering events.

Aristov et al (1991) describe a method for tabulating the output of a Monte Carlo simulation by considering the number of times an electron crosses the plane $z=0$ if it is scattering in an infinite volume starting from the origin in a direction normal to that plane. While potentially very fast, this method is limited to surfaces that are essentially flat at the scale of the interaction volume (the volume that contains most of the scattering electrons before they run out of energy or escape from the sample). It would be interesting to see if this sort of model (handling small-scale topography) could be combined with a slope-based model. Most other Monte-Carlo approaches replace some subset of interactions with an approximation. Plural-scattering methods (as opposed to single-scattering) as used by Joy, Ly and others simplify the calculations by considering the net effect of multiple scattering events. Mixed simulation models as used by Acosta (1996) do fully detailed calculations for the more critical scattering events (large angle or elastic scattering) that are most important for determining the shape of the interaction volume while condensing the more common but less critical events such as SE generation and other inelastic interactions. Single-scattering models as used in MONSEL and several other programs attempt to model all elastic collisions and those inelastic scattering events most important for SE generation. If some of the
inelastic scattering is not taken into account, a continuous energy loss formula is commonly used to estimate the energy of an electron as it slows down in the solid. The energy loss formula is usually based on a combination of theory and experiment and the one used in MONSEL is notable for its reasonable behavior at very low energies\textsuperscript{24}. The most detailed models as developed by Shimizu et al (1976) avoid the use of a continuous energy loss formula but such models require data on the sample material that is not available for most materials\textsuperscript{36}.

For this work, the MONSEL model was used because it is accurate and includes fitting parameters to match experimental data. In particular, MONSEL provides a realistic simulation of secondary electron emission that is not handled by many of the other simulators\textsuperscript{24}.

In MONSEL, the elastic scattering cross section is computed using the empirical fit by Browning et al (1994) to the Mott cross-section\textsuperscript{4}. Secondary electron generation through ionization of valence electrons is computed using the Møller cross section described in Reimer (1985)\textsuperscript{33}. Secondary electron generation through plasmon excitation is handled using the model of Kotera et al (1990)\textsuperscript{20}. When an inelastic scattering event is simulated, the energy of the resulting secondary electron is not subtracted from the primary electron. Instead, the expected loss is accounted for using a continuous energy loss model\textsuperscript{24}.

Simulation design

We used two optimizations to reduce computation while leaving the accuracy unaffected. The first optimization is to precompute and store electron trajectories in such a way that they can be reused later for an arbitrary sample. During simulation of an image, using precomputed trajectories lets us avoid complex calculations for determining probabilities of various scattering events and allows us to focus more computing resources on determining intersections of the electron trajectories with sample surfaces. The second optimization is to precompute intersections of the precomputed trajectories with various representative surfaces. Using a relatively small set of representative surfaces and the precomputed intersection data, we are able to determine intersections for a much larger set of surfaces during image simulation. By matching each relevant surface patch on the sample with one of the representative surfaces, we are able to reuse the precomputed intersection data to compute intersections with the surface patch. We essentially share computation between parts of the sample that have similar topography.

Orthographic (parallel) image projection and detector characteristics

Our simulator assumes an orthographic projection, which means that the incident electron beam comes from the same direction for all pixels in the image. While this is not a critical design decision, it does facilitate reuse of scattering results from one part of the surface to another. If a perspective projection were used, the precomputed trajectories would need to be rotated separately for each point on the surface. Given how close the real projection is to being orthographic this seems like a reasonable approximation.

We do not attempt to model detector characteristics in any detail but we do make certain assumptions about electrons after they escape from the sample. If an electron escapes with energy greater than 50eV then it is assumed to follow a straight path in the vacuum and may reenter the sample depending on its direction. An electron escaping with less than 50eV is assumed to follow a curved trajectory in the vacuum such that it is collected by the detector regardless of its direction. If one of the higher energy electrons escapes without reentering the sample it is counted as a BSE and if one of the lower energy electrons escapes it is counted as an SE.

Storage of precomputed trajectories

We make the optimization of precomputing trajectories in an infinite volume of material and then retracing those trajectories at each point on the surface. The only difference between
trajectories that are computed in an infinite volume and those computed with a surface is that the ones for the surface have segments of vacuum inserted where the electron has escaped and reentered and terminate prematurely if the electron escapes completely from the sample. An example of a precomputed trajectory and its corresponding retraced trajectory for a particular surface is shown in Figure 4.

![Figure 4: Example of precalculated trajectory (left) and its corresponding retraced trajectory (right). A section of the trajectory traversing the vacuum is shown with a dotted gray and the point of escape from the sample is shown with a dotted gray arrow.](image)

As long as an electron does not cross a material boundary, the same precalculated trajectory can continue to be used for that electron. If a material boundary is crossed then the program computes the remaining part of the trajectory from scratch. Since material boundary crossings are expected to be relatively rare for most applications we do not expect them to significantly slow down the program in the average case.

If one were to retrace the same set of trajectories for every pixel in an image, the resulting image would contain considerable bias. In order to produce a less biased result, a set of 10000 trajectories was precomputed. For each pixel a random subset containing 1000 of the 10000 precomputed trajectories was retraced to determine the BSE and SE yields for that pixel.

**Precomputation of intersections with approximating surfaces**

This algorithm for accelerating the simulation, illustrated in Figure 5, is only possible because we are using precomputed trajectories. To start, a bounding box is constructed that contains all precomputed trajectories. This bounding box is used to define a set of neighborhoods (one neighborhood per pixel in the image) containing each beam entry point on the surface. For a given pixel, when a beam electron enters the surface it is guaranteed not to escape the surface outside of the corresponding neighborhood. Next, the surface patches corresponding to the neighborhoods are clustered into a relatively small number of sets according to slope (the sets are labeled A, B and C in Figure 5). For each set, a plane is defined that has a normal equal to the average for the set and with a height such that it bounds all neighborhood surfaces in the set from below. Next, for every precomputed trajectory and for every plane approximation, the parts of the trajectory lying above the plane are computed and stored.

During retrace of a trajectory for the actual surface the program uses the precomputed intersection information to skip over those parts of the trajectory lying under the corresponding approximating plane. The program only performs this optimization up until the first escape of an electron from the surface (whether it reenters or not) or material boundary crossing because the precomputed results are no longer applicable after such an event. While this offers no benefit for those parts of the trajectory that must be traced after an initial escape, we expect most of the tracing to occur before any escape. The benefits of this optimization depend on the self-similarity of the sample surface. Although this acceleration technique is applicable to almost all sample models consisting in some part of bulk material, the fact that electrons can only escape from an upward-facing surface in the case of a height field simplifies some calculations.
Parallelization

To speed up the simulation, both the precomputation of trajectories and the retracing of those trajectories were performed in parallel on up to 30 processors. Each processor computed a complete image using a subset of the trajectories and the resulting images were added together to get the final result.

Object-oriented design and extensibility

Two parts of the system that are worth noting from a software engineering standpoint are the basic electron scattering model and the sample model. Both of these parts were designed to be very extensible. This flexibility enables one to easily adapt the simulation to use a different model for electron scattering or the sample.

The scattering model is constructed in a modular fashion from any number of C++ objects describing various events that can occur. In our case, we have three different event objects for the various parts of our scattering model: elastic scattering model based on Mott cross section, Koter’s inelastic scattering model for plasmon excitation, and Möller’s model for valence electron ionization. Each event object is responsible for describing the total and differential scattering cross sections (cross section area is roughly proportional to the probability of a scattering event occurring within a small length of a trajectory and differential cross section is the derivative of this probability with respect to the scattering angle) for a specific type of event including resulting products such as secondary electrons or x-rays. Events may be combined using a composite event object to form a single event that specifies everything that can happen to an electron in a given material. Different component scattering events may be from different models or may be the same model for different elements that compose the sample material. Another part of the scattering model is a continuous process that may be used to describe changes in the electron energy over distance. The programmer has the option to use the continuous process object for a continuous energy loss approximation or the continuous process can be left out entirely if everything is to be handled through the event mechanism.

The parts of the program responsible for tracing electron trajectories make no assumptions about what sample model or representation is used. The only requirement is that the material at every point in space is well defined. Testing the intersection of trajectories with boundaries is handled through a function pointer mechanism. The function specifying surface and material boundaries must take a ray or ray segment and return whether or not that ray intersects a boundary and if so it must also return the location of the intersection and what material if any is on the other side of the boundary.

Performance measurements

Several tests were run to compare the performance of our program with that of CASINO$^{11}$ and the original MONSEL Fortran code.

Comparison with CASINO

This test compared versions of our program with and without trajectory precomputation and precomputed intersections with the CASINO
program: CASINO our program our program + precomputed trajectories our program + precomputed trajectories and intersections

# electrons 10000 10000 1000000 1000000

time (seconds) 5 4.2 57 3.56

electrons/second 2000 2380 17543 280900

Table 1: comparison with CASINO computing backscattered electrons from flat silicon surface

program. CASINO was only available for a PC running Microsoft Windows so this test was run on a Pentium III 300 MHz PC with Windows 2000. We disabled x-ray calculations and display of electron trajectories in CASINO and disabled secondary electron calculations in our program to make the comparison as fair as possible. We also matched scattering models as closely as possible by making CASINO use the Browning fit to the Mott partial and total cross sections and making both programs terminate trajectories at a minimum energy of 0.05 keV.

The test consisted of computing trajectories for 10000 1 keV electrons striking a flat silicon surface. The results are given in Table 1.

**Comparison with original MONSEL Fortran code**

For this test we had both programs compute 10000 1keV electrons hitting a flat Silicon surface as in the previous test but with calculations of secondary electrons enabled. This test was run on an SGI reality monster with 32 processors but the programs were run on a single processor (300 MHz IP27 MIPS R12000 CPU with R12010 FPU). The results are given in Table 2.

The measurements of our program using precomputed trajectories and intersections give a sense of the best-case performance possible when simulating an image of a more complex surface. In practice we found that the additional calculations required when a back-scattered electron escapes and reenters the sample and the imperfect fit of precomputed surfaces to the simulated surface greatly reduced the effectiveness of the precomputed intersection technique. An improvement that we leave for future work is the extension of the precomputed intersection technique or an alternative method to help with the case when a backscattered electron reenters the surface.

| program: MONSEL (F77 Fortran) our program our program + 10000 precomputed trajectories our program + 10000 precomputed trajectories and precomputed intersections with a flat plane |
| --- | --- | --- | --- | --- |
| # electrons | 10000 | 10000 | 10000 | 10000 |
| time (seconds) | 24.7 | 23.2 | 4.21 | 0.122 |
| electrons/second | 405 | 431 | 2375 | 81970 |
| memory used | -- | 3 MB | 397 MB | 410 MB |
| # trajectory segments traversed | -- | 3872277 | 3884425 | 6184 |

Table 2: comparison with original MONSEL Fortran code showing performance gain with precomputed trajectories and the best case performance using precomputed intersections
**Speedup using multiple processors**

These tests were done on an SGI reality monster with 32 300 MHz MIPS R12000 CPUs and 16384 Mbytes of main memory.

In the first test using multiple processors, images were simulated of a very simple model, an infinite plane, to get a sense of the best-case performance. This test was done using 10000 precomputed trajectories for 1keV electrons hitting Silicon, precomputed intersections, 1000 retracted trajectories per pixel, and 256x256 pixels. A single processor took 754.7 seconds (86837 electrons/second) to do this computation while 30 processors running in parallel took 27.9 seconds (2,325,000 electrons/second). The speedup for various numbers of processors is shown in Figure 6.

![Figure 6: Speedup as a function of number of processors computing an image of an infinite flat Silicon surface using 10000 precomputed trajectories and precomputed intersections.](image)

A similar test was performed for a much more complex polygonal surface model composed of 79,202 triangles shown in Figure 7a.

In this case 1000 trajectories were simulated for every pixel using 10000 precomputed trajectories and precomputed intersections. While the surface is 398 nm across, the view in the simulated images is limited to a square region in the center that is 192 nm across. The images generated are shown in Figure 7(d-e) and the speedup for various numbers of processors is shown in Figure 8. A single processor took 1620 seconds (2528 electrons/sec) while 30 processors running in parallel took about 93 seconds (43600 electrons/second) to do this simulation. We are not sure why there is a reduction in efficiency for larger numbers or processors but it might have something to do with a change in patterns of memory access and the fact that the whole surface is accessed by each program thread. It would be interesting to see if this problem occurs with the same geometry represented more compactly so it wouldn’t be expected to have such a strong effect on memory use. In our SGI machine memory is divided into separate modules each shared by two processors. With only a single copy of the surface

![Figure 7: (a)Triangle mesh representing a sample surface. The surface is 398nm x 398nm in x and y and the square projection is 200 nm high. (b) Image showing height of the surface for each pixel in the simulated images (region is 192nm x 192 nm). (c) Map of pixels for which precomputed intersections are used (grey indicates where they are not used). (d) Back-scattered electron image. (e) Secondary electron image.](image)
in memory and the fact that it won’t fit in the relatively tiny data cache each processor must access the surface data over the internal network introducing delays into the intersection calculations. Therefore, one possible solution would be to have each processor create its own copy of the surface, forcing the surface to be replicated in each memory module. It is interesting to note that the same problem can not occur with memory access of precomputed trajectory data because this is done completely separately by each thread of the program each with its own unique set of trajectories. If we had chosen to divide the work by pixels instead of by trajectories, it would require each processor to access the trajectory data over the network. It would be much less practical to replicate the trajectories per processor because each 1keV trajectory takes about 50kBytes (500MBytes for 10000 such trajectories). Also, at 1keV precomputed intersections require a significant amount of additional memory when including secondary electrons because of the additional information that must be recorded each time a trajectory branches: 1600bytes/trajectory/plane (leaving out secondary electrons reduces the memory requirement to 13bytes/trajectory/plane). By contrast, our surface representation requires about 375 bytes per triangle or about 30MB for a surface specified with a 200x200 grid of points.

Simulated SEM images were created based on topography measured with an AFM. In this case we took measurements of a surface composed only of silicon that had been etched with an interesting grid pattern. This sample was provided by John Dagata at NIST. The AFM topography is shown as a 3D rendering in Figure 9.

![Figure 9: 3D graphics rendering of surface acquired by an AFM. The surface is 3000 nm x 3000 nm and 260 nm high and composed of silicon. It is represented as a triangle mesh with 79202 triangles.](image)

The 600x600 pixel image back-scattered electron image shown in Figure 11 was generated using 16 processors on the SGI reality monster in 18 min 43 seconds at a rate of 19870 electrons/second/processor. This simulation included the precomputation of 10000 electron trajectories and retracing a random subset of 992 of these trajectories for each pixel in the image. This simulation also made use of precomputed intersection calculations for about 70% of the pixels.

![Figure 8: Speedup as a function of number of processors computing an image of a polygonal model with 79202 triangles](image)

Simulation of measured surface topography
a test, we used a complex model of a dragonfly composed of 7570 triangular faces.

The resulting 512x512 pixel simulated images shown in Figure 13 and Figure 14 are particularly interesting because they demonstrate the volumetric nature of the SEM measurement where electrons have passed through the thin wings and hit the body.

The material was modeled as silicon and the scale was chosen such that the body is approximately 800 nm long and the wings are approximately 1.44 nm thick. The ray-object intersection tests used for this simulation were far from optimal so this simulation ran quite a bit slower computing only about 66 electrons/sec/processor (not including electrons not hitting the object) for a total time of about 30 minutes when 100 1keV precomputed incident electron trajectories were retraced for each pixel on 24 processors. In this case in contrast with the surface example, a significant number of pixels don’t require computation of electron scattering because the corresponding incident rays don’t hit the object.

Simulation of arbitrary polygonal models

To demonstrate the generality of our software, we implemented a simple ray intersection routine for a general polygonal model of a solid sample. As
Acknowledgements

The authors are grateful to John Dagata for supplying the sample used to generate Figure 9, Figure 10 and Figure 11 and to John Villarrubia for providing us with some basic source code from MONSEL.

References:

   http://www.gel.usherbro.ca/casino/.