

Illustrating Magnetic Field Lines using a Discrete Particle Model

Thomas Klein and Thomas Ertl

Institute of Visualization and Interactive Systems, University of Stuttgart
Universitätsstraße 38, 70569 Stuttgart, Germany
Email: {klein, ertl}@vis.uni-stuttgart.de

Abstract

In this paper we demonstrate how a computational physics approach based on a discrete particle model can be employed for the visualization of magnetic field lines. This can be regarded as an illustration technique for the underlying vector field. To accomplish this, a model for ellipsoidal shaped particles consisting of ferromagnetic materials is described in combination with a parallel simulation algorithm based on an electrostatically enhanced Molecular Dynamics approach. For interactive viewing of the time-dependent simulation data, a hardware-based projection algorithm for the perspective correct rendering of large numbers of illuminated ellipsoids is presented.

1 Introduction

Experimental visualization of magnetic field lines is a well-known technique. Probably no student leaves school without having seen simple demonstrations carried out with a permanent magnet and some iron filings. With this simple and yet powerful experiment it is possible to gain much insight into the structure and properties of magnetic fields. So why not also use this method most people are familiar with in a computer graphics tool to illustrate the shape of magnetic fields, e.g. in education or in popular scientific publications? This could be especially useful, if real experiments are not possible due to a complicate experiment setup or large scale fields which can not be reproduced in a small scale experiment are involved.

In this paper we present a physically based simulation method for the dynamic behavior of particles of ferromagnetic materials in an applied magnetic field. Using a full three-dimensional model allows for the simulation of a wide range of phenomena that can be observed during the alignment process

of the particles, e.g. particles that pile up or are restricted in motion due to their shape. It is important to model several physical properties and laws that are involved. First, there are the mechanical properties of the particles, e.g. their size and shape and material properties like friction coefficients or elasticity, that, in coaction with for example gravity or obstacle geometry, influence the motion of the particles and their interaction with adjacent particles. As large numbers of finite-sized particles have to be used in order to produce illustrations that resemble the appearance of real world experiments, typical rigid body simulation approaches known from computer animation do not suffice. Instead, methods from computational physics that deal with large many-particle systems have to be used. For the magnetostatic interaction between particles and between particles and an external magnetic field, two additional effects have to be incorporated: The magnetization of a particle due to the given external magnetic field and the induced magnetic fields of the other particles, and the forces and torques acting on the particles as a result of that.

Furthermore, since for a reasonable simulation the size of the problem, given by the number of discrete particles, can easily exceed 100,000 interacting objects, this computational task is only feasible when high performance computers like large shared memory machines or cluster computers are used. In this work a MPI-based [2] implementation is presented that allows us to simulate large numbers of particles while moderate computing times are achieved.

2 Related Work

Traditionally, the visualization of magnetic fields or more general vector fields is done via field lines. But there are also many problems associated with this approach. How to find appropriate seed points?

How to achieve a reasonable coverage or line density, especially in 3D? Many lines have to be pre-computed in order to visualize the whole field. Another popular method, known as Line Integral Convolution (LIC), that addresses at least some of those limitations in a way that it produces a dense representation of the whole field, has been the ancestor of an entire family of modern flow visualization methods, cf. [9] for an overview.

Although, the magnetic field is a vector field, it is in fact no flow field. Therefore conventional LIC is not very well suited for visualizing magnetic fields. An improved method that was specifically designed for vector fields that exhibit no typical flow behavior such as electric fields, was presented by Sundquist [14]. This method, called Dynamic Line Integral Convolution (DLIC), addresses the fact that in electromagnetic fields, unlike flow fields, the direction of the motion is not necessarily the direction of the field itself.

The method we want to present in this paper, in turn, is not meant to be an alternative to the above mentioned algorithms for interactive visualization in scientific applications, but rather as a way to generate near photo-realistic illustrations of magnetic fields that can be used for example in educational or instructional contexts.

The dynamic behavior of particles in magnetic fields is still subject of current research in computational physics. Most of the published papers regarding the simulation of the behavior of granular magnetizable materials focus on the investigation and improvement of the compaction process in manufacturing of magnetic materials such as permanent magnets or magnetic recording media [7].

3 Simulation of Particles in a Magnetic Field

In order to achieve realistic and visually pleasing illustrations large numbers of particles have to be used. From our experiments we found that, depending on the underlying magnetic field and the resolution of the generated images, at least 30,000 to 50,000 particles should be used. Although, for some of our experiments we used up to 100,000 particles to achieve satisfying results. For such a large number of distinct objects the well-known methods traditionally used in computer animation are no longer feasible. Instead we have to use methods

from computational physics where dynamic many-particle systems consisting of several thousands to millions of particles have attracted the attention of researchers for more than 50 years. There are several well-known and widely used methods available that could be applied to our problem. We have chosen a Molecular Dynamics approach that is known to deliver realistic results, takes the geometric relationship into account, which is crucial in a graphics application, and is also easy to parallelize.

Molecular Dynamics (MD) is a common technique for the simulation of large numbers of distinct particles, e.g. in the study of atomic clusters, in biomolecular modeling, the simulation of liquids or granular matter. Each molecular object, e.g. atom or small molecule, is represented by a single particle or mass point that interacts with other particles and its environment via pairwise potentials. In basic MD the evolution in time, i.e. the trajectories, of the particles is computed by integrating the Newtonian equations of rigid body motion depending on forces derived from the given interaction potential and, maybe, external forces. Whereby the actual potential has to be deduced from certain physical laws, e.g. van der Waals forces. This simple microscopic model can be extended to a macroscopic model that can handle the interaction of particles composed of millions of molecules. For such a model, obviously, a simple potential function is not sufficient to describe the interaction between two particles. Therefore, some kind of geometric particle-particle interaction potential has to be defined. For two elastic particles that cannot interpenetrate each other, the expected qualitative behavior of the contact potential would be as follows: Neglecting adhesion, the two spheres do not exert any force on each other as long as their surfaces do not touch, but as soon as they get in contact with each other the balls will deform and a repulsive force will arise between them. The greater the deformation, the greater the repulsive force.

In traditional rigid body collision systems a hard object model is used, in which collisions are treated by applying linear momentum at the time of contact between two objects and restarting the time integration. Thus, for many tightly packed objects, the effective time step tends to get very small. In MD a constant time step is used. That means, instead of enforcing the constraint that particles can not penetrate each other, the assumption that the two col-

liding particles undergo a slight elastic deformation is introduced. In this, so-called *soft particle* model, the overlap, that occurs due to the fact that essentially the particle model does not reflect those virtual deformations, provides a measure for the repulsive force and is called *virtual overlap*. The resulting repulsion force is typically modeled by the behavior of a damped nonlinear spring acting in the direction of the collision normal. Additionally to these normal forces, e.g. frictional forces for grazing contacts can be modeled.

3.1 Particle Model

Most particle dynamics codes still use a very simple, spherical particle model. In this work, instead, general ellipsoidal particles, i.e. ellipsoids with no restrictions concerning the lengths and ratio of the three semi-axes, are used. There are several reasons why we use ellipsoidal particles for the simulation. First of all, it provides us with a reasonable good approximation to real particle shapes, such as the iron filings we want to simulate. Although more elaborate particle models, e.g. super-ellipsoidal forms [16], multi-element approaches [3] or polygonal approximations [11], have been described that may provide an even better coincidence with the shape of real particles, the use of ellipsoids has some additional advantages. They can be analytically described in a very simple way, but nevertheless provide a wide range of different particle shapes, ranging from spherical over elongated or needle-like forms to oblate or disc-shaped objects. Furthermore, the ellipsoidal shape exhibits certain magnetic properties that are important for the computation of the particle magnetization. Ferromagnetic samples are more easily magnetized in the direction of greater extent than in others, causing an anisotropy of the magnetization. Therefore, spherical particles are not well suited for our purpose.

In order to perform a MD simulation, a method for finding particle-particle contacts and computing the overlap between two general ellipsoids has to be introduced. Unlike the spherical case, determining if two general ellipsoids intersect or are separated from each other is no trivial task. There are several methods, e.g. [18], that have been proposed for finding the intersection curve of two ellipsoids in three-dimensional space. But as we are only interested in the fact whether there is an in-

tersection and if this is the case how large the actual overlap is, our problem is slightly different. A special contact function for ellipsoids that accomplishes that task was proposed by Perram and Wertheim [13] for Monte-Carlo simulations. Other algorithms for contact detection and overlap computation in discrete element simulations are based on solving quartic equations [15] or a geometric potential concept [10]. The drawback common to all these methods is that they all rely on numerical iteration techniques for either solving the resulting differential equations, finding roots or minimizing functions.

In order to improve the robustness and the speed of the initial collision test we employ an algebraic criterion for the separation of two ellipsoids [17]. It requires no iterative computations, and thus is more efficient and exact compared to the aforementioned methods. In [17] it is shown that two ellipsoids of the form $G_k(\mathbf{x}) = \mathbf{x}^t A_k \mathbf{x} = 0$ have an intersection if and only if their characteristic equation $f(\lambda) := \det(\lambda A_i + A_j)$ has at most one distinct positive root. Given that only the signs of the roots are important, the task of deciding whether the ellipsoids intersect can be accomplished without explicitly finding those roots [6].

If a contact was detected, the virtual overlap of the ellipsoid pair has to be computed. This is done with an iterative method based on a steepest descent approach that finds the two points of deepest penetration, hereafter referred to as the *contact points*. Fig. 1 shows a simplified conceptual view of the algorithm. Starting with an initial guess on the boundary of the first ellipsoid $G_1(\mathbf{x}) = 0$, walking tangentially along that boundary in a way that minimizes the value of the function $G_2(\mathbf{x})$ the first contact point is found. The descending direction is chosen depending on the gradients of the two quadratic forms. The locally strongest minimization is to be expected in the direction of the negative gradient $-\nabla G_2(\mathbf{x})$. But as this gradient is not mandatory tangential to the surface of the first ellipsoid, it has to be projected into the tangential plane in order to find \mathbf{d}_i , the direction of interest. For a given approximation \mathbf{q}_i , the next point \mathbf{q}_{i+1} is found by walking along the straight line $\mathbf{q}_i + \lambda \mathbf{d}_i$ with step size $h > 0$ and projecting the point $\mathbf{q}_{i+1}^* = \mathbf{q}_i + h \mathbf{d}_i$ back onto the surface $G_1(\mathbf{x}) = 0$. The contact point \mathbf{q}_c is reached as soon as $\nabla G_1(\mathbf{x}) \parallel \nabla G_2(\mathbf{x})$, which is a reasonable assumption for small overlaps. Thereby

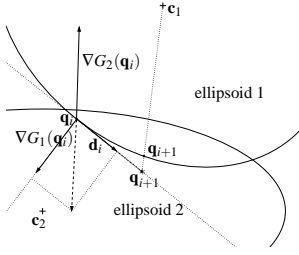


Figure 1: Finding the contact point.

special care has to be taken about the “leaping over the contact point” that can occur if the step size h is chosen too large. In this case, adapting the step size to $h := h/2$ ensures convergence. The virtual overlap is then given by the Euclidean distance of the two contact points.

3.2 Magnetic Interaction

We use an extremely simplified but still physically based model for the magnetic properties and interactions of our particles. First of all we assume that each particle can be represented by a single macroscopic magnetic dipole $\mathbf{m} = V\mathbf{M}$ according to its current magnetization vector \mathbf{M} and its volume V . Then, computing the magnetic interaction consists of two main aspects: magnetization of the particles and forces and torques acting on the particles due to the magnetic field they are exposed to. This magnetic field consists, on the one hand, of the applied external magnetic field \mathbf{H}_{ext} and, on the other hand, of the secondary induced fields $\mathbf{H}_j = \mathbf{B}_j/\mu_0$ of all magnetized particles.

First, the forces and torques acting on the particles due to the magnetic field they are exposed to have to be considered. These are given by

$$\mathbf{F} = \nabla(\mathbf{m} \cdot \mathbf{B}) \quad \text{and} \quad \boldsymbol{\tau} = \mathbf{m} \times \mathbf{B},$$

where \mathbf{B} is the induction of the magnetic field given by the superposition of \mathbf{H}_{ext} and the dipole fields \mathbf{H}_j of all other particles. This can be written as an additional pairwise particle-particle interaction term for the MD simulation.

Computing the particle magnetization requires some more effort. We use a micromagnetic approach that reduces the problem to a multi-dimensional minimization problem of the total free magnetic energy. According to [8] this energy of a

ferromagnetic particle in a magnetic field is given by

$$E = E_z + E_s + E_c + E_e + E_\sigma, \quad (1)$$

where E_z denotes the energy of the sample due to the torque the external field $\mathbf{H} = \mathbf{H}_{\text{ext}} + \sum \mathbf{H}_j$ exerts on the elementary dipoles (Zeeman energy), E_s represents the energy of the particle in its own magnetic stray field, E_c and E_e the magnetocrystalline anisotropy energy that arises from the alignment of the elementary dipoles with respect to the crystalline structure of the material and the exchange energy that originates from the interaction of the elementary dipoles, respectively. The last term E_σ denotes the magnetic stress anisotropy energy that is caused by stress exerted on the samples crystalline structure.

We simplify this expression by introducing some more assumptions. First, we assume that the particles we deal with are quite small, i.e. they consist of only one magnetic domain. Thus, the exchange energy can be neglected. We also neglect E_c and no stress is taken into account. Last, we assume that the particle magnetization is always saturated, i.e. $|\mathbf{M}| = M_s$. With these assumptions the total energy reduces to $E = E_z + E_s$. Only two terms are left: The Zeeman energy and the self-energy of the particle in its stray field \mathbf{H}_s . As \mathbf{H}_s is always acting against the particle’s magnetization it is also called *demagnetization field*. This field introduces an anisotropy depending on the shape of the sample. In general, the demagnetization tensor N that characterizes the shape anisotropy and, therefore, E_s are very complicated to compute. But, due to the simple symmetries of the ellipsoidal shape the demagnetization field inside the ellipsoid is homogeneous and the demagnetization tensor is diagonal and its elements can be computed very easily [12]. Thus, the magnetic energy of the i th particle can be expressed in terms of the magnetization vectors \mathbf{M}_j of all particles as follows:

$$E_i = -\mu_0 V_i \mathbf{M}_i \cdot \mathbf{H}_i + \frac{1}{2} \mu_0 M_s^2 V_i \sum_{k=1}^3 N_{i,k} \alpha_{i,k}^2 \quad (2)$$

with

$$\mathbf{H}_i = \sum_{j \neq i} \frac{1}{\mu_0} \mathbf{B}_j = \frac{1}{4\pi} \sum_{j \neq i} V_j \left(\frac{3(\mathbf{M}_j \cdot \mathbf{r}_{ij}) \mathbf{r}_{ij}}{|\mathbf{r}_{ij}|^5} - \frac{\mathbf{M}_j}{|\mathbf{r}_{ij}|^3} \right),$$

and the direction cosines $\alpha_{i,k}$ of \mathbf{M}_i with respect to the half-axes vectors \mathbf{h}_k of the ellipsoid. This

yields a system of non-linear equations that have to be minimized simultaneously for the directions of the n unknown vectors \mathbf{M}_i .

$$E(\mathbf{M}_1, \dots, \mathbf{M}_n) \rightarrow \min. \quad (3)$$

Finding a minimum for this systems would be very time-consuming for a large number of particles. Therefore, we further assume that for each particle \mathbf{M}_j is constant for all $j \neq i$ during the minimization, thus linearizing and decoupling the system. This allows us to solve for the directions \mathbf{M}_i independently by means of a conjugated gradient iteration method.

3.3 Parallel Implementation

The complete MD simulation algorithm is implemented in C++ using MPI [2] to distribute the computation over the nodes of a PC cluster. A cell-based space partitioning scheme typically referred to as the linked-cell-list approach is used to speed up the computation of the contact forces. In contrast to the short-range contact potential the magnetic interactions are not limited to the vicinity of a particle, i.e. each particle interacts with all other particles in the simulation area. Therefore, a simple approach as used for the mechanical particle-particle interaction is not sufficient. Although, there have been many different methods proposed that speed up long-range interaction computations in many-particle simulations, these techniques are either only applicable to periodic systems or rather complex to implement. Therefore, we have chosen a simplified version of a PPPM technique (Particle-Particle Particle-Mesh) proposed in [4] that is quite similar to the linked-cell method used for the short-range interactions. It is based on the following observation: The strength of the magnetic dipole field of a particle varies with the inverse of the cube of the distance from the particle's center and the respective forces arising between particles decrease even faster. Taking into account that all particles are of approximately the same size and assuming that they are relatively evenly distributed in space it is possible to use the spatial partitioning to speed up the computation of the magnetic interaction, as well. The direct particle-particle interaction is computed only for the particles inside the same cell and the 26 neighbor cells. The contributions of all other particles are approximated on a per cell basis. First, for each cell a "virtual" magnetic dipole is introduced that is defined by the center of mass of the

particles in the cell and the sum of their single magnetic moments. Then, for each cell center the magnetic field generated by all other cells—excluding the cell itself and its neighbors—is computed from those "virtual" dipoles. This gives us an approximation to the field of the farther away particles. Thus, \mathbf{H}_i can be approximated as

$$\mathbf{H}_i \approx \sum_{j \in \Omega \setminus i} \frac{1}{\mu_0} \mathbf{B}_j + \mathbf{H}_{\text{grid}}, \quad (4)$$

with Ω being the set of indices of the particles contained in the cell the i th particle is located in and all of its 26 neighbor cells.

To achieve the maximum parallelism, an evenly distributed load for all used cluster nodes is necessary. Therefore, dynamic load balancing using graph partitioning functions from the ParMetis library [1] is used. The cells of the space partitioning are assigned to the cluster nodes according to the partitioning computed with a k-way partitioning algorithm from the weighted dual graph of the grid cell structure. The vertices and edges of the graph are weighted depending on the computational load, i.e. the number of particle-particle computations that have to be carried out for this cell, and the communicational effort, i.e the amount of data that has to be transferred potentially from one compute node to another if the two neighboring cells are located on different nodes.

4 Particle Rendering

During the particle simulation step large amounts of time-dependent simulation data are produced. For most of the examples we have computed, 50,000 particles or more were used and the simulation results in more than 1,000 saved time steps. That means several gigabytes of total simulation data have to be handled. Rendering that many particles at interactive rates becomes tedious when a sufficiently smooth geometric representation of the ellipsoids is required, due to the large amount of geometry that is involved. Even when OpenGL display lists or vertex arrays are employed the frame rates that can be achieved for reasonably smooth tessellated ellipsoids are far from being interactive. Therefore, we have come up with a solution that breaks down the generated geometric data per rendered ellipsoid to a single vertex position with two additional per-vertex attributes. The basic idea is to

render just a single OpenGL point per ellipsoid and exploit the programmability of both the vertex and the fragment processing pipeline of recent graphics processing units (GPUs) in order to compute the perspectively correct projection of a Phong-shaded ellipsoid directly on the graphics board, thus avoiding the bottleneck of the AGP bus in exchange of higher vertex processing and rasterization effort. As only one extended point primitive is rendered for each ellipsoid our approach could be also thought of as a kind of splatting method.

Our approach is very similar to one recently presented by Gumhold [5]. But our method is also different from this method in that we reduce the amount of information that has to be transferred over the AGP bus to the necessary minimum. Each ellipsoid is represented by its center point \mathbf{c} , a half-axis vector \mathbf{h} and a quaternion \mathbf{q} describing its orientation. These three values can be encoded into the 3D vertex position, the normal vector and the four element vertex color attribute of a single `GL_POINT` primitive. Thus, only ten floating point values, or 40 bytes, per ellipsoid have to be transferred over the system bus to the GPU.

The rough outline of the algorithm is as follows: First, for each ellipsoid the screen area that is covered by its perspective projection is determined. Then, the fragments that actually lie inside the silhouette of the projection have to be determined. For each pixel covered by the point primitive a ray from the eye-point through that pixel is computed. This eye-ray is intersected with an implicit representation of the ellipsoid in order to decide if the fragment belongs to the projected boundary or not. Depending on this the fragment is either discarded or the intersection point and the ellipsoid normal are computed for shading the resulting pixel.

Associated with each ellipsoid there is a local coordinate system in which its boundary can be represented implicitly by the quadratic form

$$\left\{ \tilde{\mathbf{x}} \mid \|\tilde{\mathbf{x}}\|^2 = 1 \right\}. \quad (5)$$

Starting from Eqn. (5) the world space description

$$\left\{ \mathbf{x} \mid \|M^{-1}(\mathbf{x} - \mathbf{c})\|^2 = 1 \right\} \quad (6)$$

of the ellipsoid can be derived, where $M = QH$ is a symmetric, positive definite matrix given by the scaling matrix $H = \text{diag}(\mathbf{h})$ and the rotation matrix $Q = R(\mathbf{q})$ that describes the orientation with respect to the local coordinate system.

4.1 Ellipsoid-Ray Intersection

In the first step of the rendering algorithm the screen area that is covered by the perspective projection of the ellipsoid's shell is found by projecting the eight corners of its bounding box into clip space. Since OpenGL points are always square, the point size is defined by the longer edge of the axis-aligned rectangle enclosing the projection. This is clearly a drawback of our approach, since many fragments have to be considered in the following steps that could be excluded if a non-axis-aligned screen-space area could be used. But in fact this is only a problem if the ellipsoid projects to a rather large, non-axis-aligned screen area.

The next step is to find those fragments of the rasterized point that indeed belong to the projection of the ellipsoid. This is done by intersecting the eye-rays, passing through each fragment, with the ellipsoid. If there is at least one intersection point the fragment belongs to the projection, otherwise it has to be discarded. Intersecting a straight line with an ellipsoid is not difficult, but even simpler in its local coordinate system. Because all transformations we have to deal with are affine it is quite obvious to transform the eye-ray into object space and then do a simple ray-sphere intersection computation.

The point $\tilde{\mathbf{e}} = M^{-1}(\mathbf{e} - \mathbf{c})$, the origin of our eye-ray in object space, is constant for all fragments of the projection, and thus, has to be computed only once per ellipsoid. In contrast, the ray direction $\tilde{\mathbf{d}}$ has to be computed for each fragment independently. Since we render only a single point it is not possible, e.g. to exploit the linear interpolation of vertex attributes during the rasterization for the computation of $\tilde{\mathbf{d}}$, as it is done in [5]. Instead we have to compute it from the current fragment position in screen space. Then the actual intersection computation is very easy. Inserting the ray equation $\tilde{\mathbf{e}} + \lambda \tilde{\mathbf{d}}$ into Eqn. (5) yields

$$\tilde{\mathbf{d}}^t \tilde{\mathbf{d}} \lambda^2 + 2\tilde{\mathbf{e}}^t \tilde{\mathbf{d}} \lambda + \tilde{\mathbf{e}}^t \tilde{\mathbf{e}} - 1 = 0, \quad (7)$$

which has to be solved for the intersection parameter λ_s .

4.2 Shading and Depth Correction

For a per-fragment correct lighting of the ellipsoid we have to compute the surface normal in the intersection point. Fortunately, the point of intersection $\tilde{\mathbf{s}} = \tilde{\mathbf{e}} + \lambda_s \tilde{\mathbf{d}}$ is identical to the normalized surface

normal. Computing per fragment Phong-lighting is now straight-forward. Last the correct depth ordering of the rendered objects is ensured. For each fragment the depth value has to be computed and written to the z-Buffer. This is done by transforming the point of intersection to world space, $\mathbf{s} = M\bar{\mathbf{s}} + \mathbf{c}$, and applying the model view and projection transforms. Perspective division and depth range mapping yields the corrected depth value for each fragment.

4.3 OpenGL Implementation

We implemented the described algorithm based on the NVIDIA (NV) as well as the ARB vertex and fragment program extensions. As mentioned before, the three properties that define an ellipsoid are specified by the attributes of a single vertex. These vertices are rendered from vertex arrays in order to achieve the maximum throughput of the AGP bus transfer. For each transferred vertex, a vertex program computes the point size and position of the OpenGL point that covers the 2D projection of the ellipsoid. Additionally, all values that are needed in the fragment program and are constant for all fragments of the respective ellipsoid are also precomputed in the vertex program and passed on as vertex attributes. Then the fragment program computes the eye-ray through the current fragment position and checks for intersection with the ellipsoid. If there is no intersection the fragment is discarded (*texkil*). Otherwise, fragment color and depth are computed according to the intersection point and the respective lighting model.

5 Results

Three different scenarios illustrate our method: The magnetic field of a bar magnet, the combined field of two solenoids, and the field excited by four current carrying conductors. Initially, in all examples the particles were randomly oriented and uniformly distributed in a thin layer above the ground plane. All images are shown on the accompanying color plate and were rendered on a NVIDIA GeForce FX 5950 based graphics card.

The first example, shown in Fig. 3, is a rather small setup. Here, only 6,000 particles were used to illustrate the field of a bar magnet positioned a short distance below a thin magnetic non-shielding

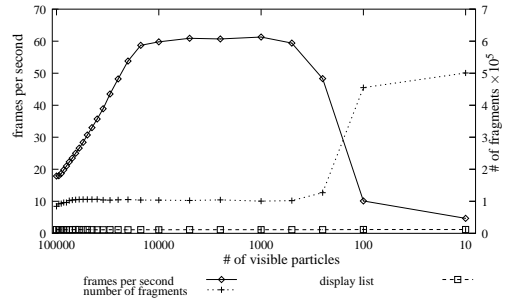


Figure 2: Rendering speed depending on the number of visible particles and written fragments².

plate, e.g. glass or paper. Nevertheless, the general structure of the underlying field is clearly visible, even for this small number of particles. On four 2.8GHz Pentium 4 PCs connected by Gigabit Ethernet the 30,000 time steps of the presented example were simulated at approximately 0.6s per simulation step. However, for more complex fields and a better visual quality, i.e. images that resemble the pictures of real-world experiments, more particles should be used. We have found that using 20,000 to 100,000 particles leads to reasonably good results.

In the second image, Fig. 4, a variation of another typical example often used in introductory texts on electromagnetism is shown: The field of two solenoids, i.e. coils with an air core. Some additional geometry is used in Fig. 4 to depict the plate, the particles are strewn on, and the coils. The simulation time for the 20,000 time steps of this example was below 0.5s per step on a eight PCs cluster.

As a last example, Fig. 5 shows a small section of a simulation with four electric conductors at different zooming levels. In this example 100,000 particles of different shapes were used. Note how the particles pile up due to the strong attraction exerted by the magnetic force in the vicinity of the (here not shown) wire. Fig. 2 shows the rendering speed for this example. Note that the written fragments are considerably less then the actual number of fragments that have to be computed in the fragment program. This is especially bad as the fragment kill instruction does not terminate a fragment program. That means if fragments are masked out because they are not contained in the ellipsoids projection, the same number of computations is carried out as for fragments that are actually written

²Pentium 4 2.8GHz, GeForceFX 5950; 512² viewport.

to the frame buffer. Moreover, if the kill instruction is used or fragment depths are written in the fragment program even the early z-buffer test is disabled. Thus, the achieved frame rate depends on the number of rendered point primitives, i.e. speed of AGP bus transfer, and the number of generated fragments, i.e. fragment processing speed. When we start to zoom in, the frame rate increases as points are culled at the view frustum. As the rendered points get larger, i.e. the projected area of the rendered particles increases, the frame rate approaches an upper limit. Then, as the number of generated fragments starts to grow very fast, since large parts of the viewport are covered by images of ellipsoids and the depth complexity is quite high, the rendering speed drops considerably. This reveals the main drawback of our algorithm: we have traded rasterization speed for bus transfer. Therefore, the proposed algorithm is best suited for settings where many particles with a relatively small projected area per particle have to be rendered while still maintaining the possibility of seamless zooming on parts of the image containing a small number of particles with a larger screen space footprint.

6 Conclusions

In this paper we have shown that a physically-based modeling technique that utilizes methods from computational physics can be successfully used for computer graphics applications. Using a Molecular Dynamics simulation algorithm, it is possible to generate illustrations of the structure of magnetic field lines by means of a discrete particle simulation. A parallel implementation on a cluster computer provides the possibility for simulating reasonably sized examples in moderate time intervals. This has been shown for practical examples of up to 100,000 particles in complex magnetic fields. In addition, a hardware-accelerated algorithm for the perspective correct rendering of illuminated ellipsoids is presented. The proposed rendering approach reduces the amount of data that has to be transferred over the AGP bus to a minimum, and thus permits the rendering of large, time-dependent simulation datasets at interactive rates. The presented results show that the illustrations that can be generated using the proposed method resemble the appearance of real world experiments with ferromagnetic powders.

References

- [1] Parmetis—Parallel Graph Partitioning and Sparse Matrix Ordering Library. <http://www-users.cs.umn.edu/~karypis/metis/parmetis/>, 2004.
- [2] The MPI Forum. <http://www.mpi-forum.org>.
- [3] J. F. Favier, M. H. Abbaspour-Fard, M. Kremmer, and A. O. Raji. Shape representation of axis-symmetrical, non-spherical particles in discrete element simulation using multi-element model particles. *International Journal for Computer-Aided Engineering*, 16(4):467–480, 1999.
- [4] H. Furukawa and K. Nishihara. Reduction in bremsstrahlung emission from hot, dense binary-ionic-mixture plasmas. *Physical Review A*, 42(6):3532–3543, 1990.
- [5] Stefan Gumhold. Splatting Illuminated Ellipsoids with Depth Correction. In T. Ertl, B. Girod, G. Greiner, H. Niemann, H.-P. Seidel, E. Steinbach, and R. Westermann, editors, *In Proceedings of Workshop on Vision, Modelling, and Visualization VMV '03*, pages 245–252. infx, 2003.
- [6] D. G. Hook and P. R. McAree. Using Sturm Sequences To Bracket Real Roots of Polynomial Equations. In Andrew Glassner, editor, *Graphics Gems I*, pages 416–423. Academic Press, 1990.
- [7] Harunori Kitahara, Hidetoshi Kotera, and Susumu Shima. 3-D Particulate Modeling for Simulation of Compaction in Magnetic Field. *IEEE Transactions on Magnetics*, 36(4):1519–1522, 2000.
- [8] Helmut Kronmüller and Manfred Fähnle. *Micromagnetism and the Microstructure of Ferromagnetic Solids*. Cambridge University Press, Cambridge, UK, 2003.
- [9] Robert S. Laramee, Helwig Hauser, Helmut Doleisch, Benjamin Vrolijk, Frits H. Post, and Daniel Weiskopf. The State of the Art in Flow Visualization: Dense and Texture-Based Techniques. *Computer Graphics Forum*, 23(2):to appear, 2004.
- [10] X. Lin and T.-T. Ng. Contact Detection Algorithms for Three-dimensional Ellipsoids in Discrete Element Modelling. *International Journal for Numerical and Analytical Methods in Geomechanics*, 19:653–659, 1995.
- [11] H.-G. Matuttis, S. Luding, and H. J. Herrmann. Discrete Element Methods for the Simulation of Dense Packings and Heaps made of Spherical and Non-Spherical Particles. *Powder Technology*, 109:278–292, 2000.
- [12] J. A. Osborne. Demagnetization Factors of the General Ellipsoid. *Physical Review*, 67(11):351–357, 1945.
- [13] John W. Perram and M. S. Wertheim. Statistical Mechanics of Hard Ellipsoids. I. Overlap Algorithm and the Contact Function. *Journal of Computational Physics*, 58:409–416, 1985.
- [14] Andreas Sundquist. Dynamic Line Integral Convolution for Visualizing Streamline Evolution. *IEEE Transactions on Visualization and Computer Graphics*, 9(3):273–282, 2003.
- [15] J. M. Ting. A Robust Algorithm for Ellipse-based Modelling of Granular Materials. *Computers and Geotechnics*, 13(3):175–186, 1992.
- [16] R. Wait. Discrete Element Models of Particle Flows. *Mathematical Modelling and Analysis*, 6(1):156–164, 2001.
- [17] Wenping Wang, Jiaye Wang, and Myung-Soo Kim. An algebraic condition for the separation of two ellipsoids. *Computer Aided Geometric Design*, 18:531–539, 2001.
- [18] I. Wilf and Y. Manor. Quadric-surface Intersection Curves: Shape and Structure. *Computer-Aided Design*, 25(10):633–643, 1993.

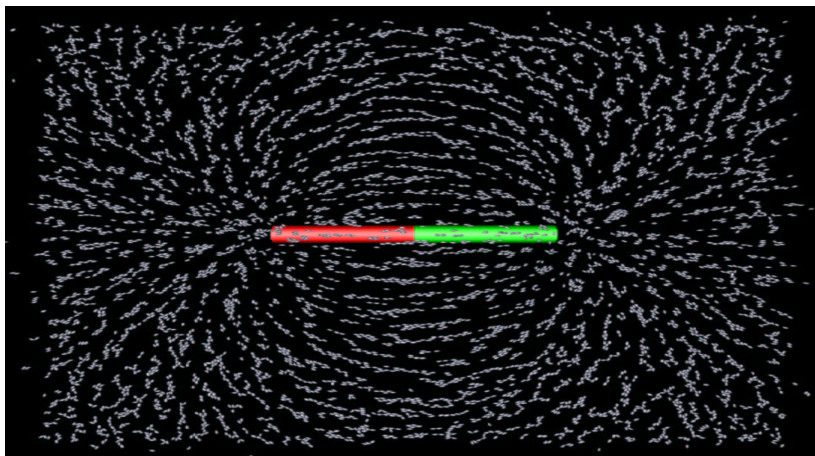


Figure 3: The field of a bar magnet illustrated with 6,000 ferromagnetic particles.

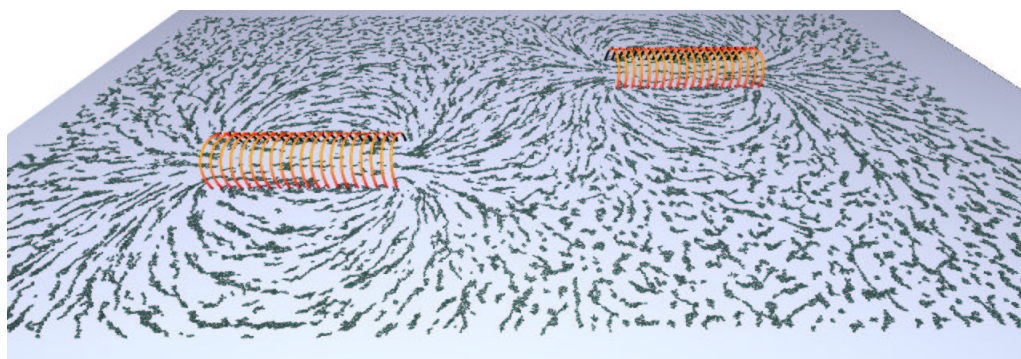


Figure 4: The field of two solenoids. 30,000 particles were used for the simulation and rendering.

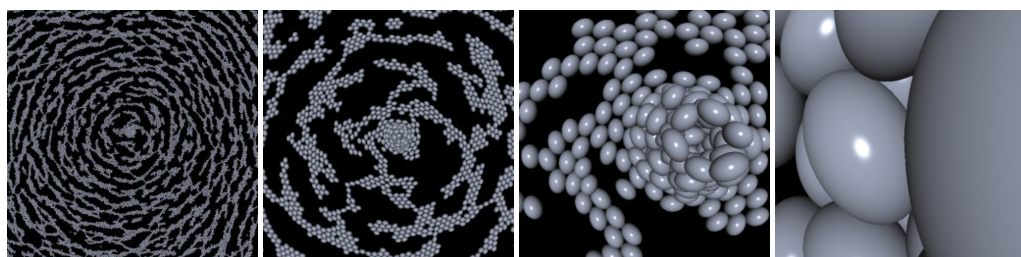


Figure 5: A small section of the result of a simulation using 100,000 particles in order to depict the field excited by the electric currents running through four conductors perpendicular to the base plane. The hardware-accelerated rendering algorithm allows interactive viewing of the time-dependent dataset, including seamless zooming in high quality.