Object orientation and visualization of physics in two dimensions

Mark Burgess, Hårek Haugerud, and Are Strandlie
Centre of Science and Technology, Faculty of Engineering, Oslo College, 0254 Oslo, Norway, and Institute of Physics, University of Oslo, P.O. Box 1048 Blindern, 0316 Oslo, Norway

(Received 18 August 1997; accepted 5 February 1998)

We present a generalized framework for cellular/lattice-based visualizations in two dimensions based on state-of-the-art computing abstractions. Our implementation takes the form of a library of reusable functions written in C++ that hide complex graphical programming issues from the user and mimic the algebraic structure of physics at the Hamiltonian level. Our toolkit is not just a graphics library but an object analysis of physical systems that disentangles separate concepts in a faithful analytical way. It could be rewritten in other languages such as Java and extended to three-dimensional systems straightforwardly. We illustrate the usefulness of our analysis with implementations of spin films (the two-dimensional XY model with and without an external magnetic field) and a model for diffusion through a triangular lattice. © *1998 American Institute of Physics*. [S0894-1866(98)01703-9]

INTRODUCTION

Although computer simulations of many physical systems are common for computing specific quantities, they do not necessarily give a direct and overall impression of the dynamics of the systems under a variety of conditions. What is often lacking from the physicist's repertoire is the possibility of a direct visualization of the behavior of microscopic systems in some appropriate semiclassical limit. Such a mental image of the mechanics of the problem can be a source of great inspiration, both for understanding established problems and for designing new scenarios. The ability to vary initial and boundary conditions (temperature, external field, and any other external parameters) and compare several simulations has enormous potential for the comprehension of the qualitative behavior, while at the same time enabling quantitative information to be calculated and displayed alongside.

In recent years the status of two-dimensional physics has changed from being a mathematical idealization to being a realistic physical prospect. The fractional quantum Hall effect and high-temperature superconductivity are widely believed to be two-dimensional phenomena to a good approximation. Moreover, developments in the growth of ultrathin layered heterostructures makes the modeling of two-dimensional systems ever more important.

Theoretical developments in two-dimensional physics have generated a large body of work. Notable topics include the notion of fractional statistics[1] or anyons (particles that interpolate between Bose–Einstein and Fermi–Dirac statistics), anyon superconductivity, and layered systems as potential models for the new high-temperature superconductors, the quantum Hall effect, and spin textures, to name but a few.[2] Many of these models could profitably be simulated visually to gain an intuitive picture of what takes place. There are many other interesting candidates for visual models in two dimensions, including the study of fields in waveguides and cavities (the micromaser) and spin computers.[3–10] Spin diodes[11] have been studied both experimentally and theoretically and could profitably be made into a visual simulation.

Recently several groups have begun to appreciate the importance of direct visualization in two-dimensional systems. Since direct experimental observations are difficult to achieve, and certainly difficult to repeat under identical conditions, computer simulations are an obvious and valuable surrogate. Computer visualization always involves some form of compromise or approximation, and sufficient control of these compromises is an important concern, but the bonus of a concrete dynamical picture of a physical system often outweighs minor qualms about their accuracy. Graphical representation of data places heuristic understanding before precision.

Certain physical situations are ideally suited to visualization. For example, any kind of field — be it of a scalar or vector character — with arbitrarily complicated boundary conditions is easily rendered visually, but might be described by complicated special functions algebraically or numerically, making it difficult to gain a qualitative understanding.

In this article, we present a visualization scheme for two-dimensional systems and layered three-dimensional systems, which may be used to play physical simulations on a lattice like a one-way video recording. Initial and boundary conditions may be edited, and parallel simulations employing different parameters may be compared. The framework is general, but we choose to illustrate it by looking at two-dimensional spin systems and diffusion models, which provide a perfect illustration of the principles.

I. CELLULAR AUTOMATON SIMULATION ENVIRONMENT (CASE)

A convenient framework on which to base a system of visualization is the cellular automaton. The idea of cellular automata was introduced around the 1970s and 1980s with

  0894-1866/98/12(3)/274/12/$15.00

key papers by, among others, Wolfram. For a review with references see Ref. 12. A cellular automaton may be thought of as a simulation engine in which cells (or lattice plaquettes), arranged in a symmetrical pattern, interact with their neighbors according to well-defined rules. Each cell or site has a number of variables associated with it, describing the state of the system at that point (e.g., temperature, occupation number, spin state, etc.), and that state evolves in time according to a supplied rule that embodies the dynamics of the model. A suitable visual representation of the automaton can be constructed by choosing a property such as color to represent temperature (or other scalar quantities) and arrows to represent vector quantities. Combinations of these, together with the ability to limit and change the variable being displayed, allow complex models to be visualized in detail.

Cellular automata have traditionally been used by physicists and biologists for studying such diverse problems as the development of cellular life, mixing of fluids, penetration of porous media, magnetic spin systems, etc. Any system in which space and time can be discretized into an appropriate lattice can be modeled by a suitably complicated automaton, and this is in tune with the unavoidable granularization that any system must undergo in a visualization. Increasingly, cellular automata are being used as simulated ''analog computers'' by engineers and statisticians. They are used in the modeling of traffic flow at junctions, electrical networks, and the diffusion of gases and spreading of interfaces, to name but a few applications. It has been recently shown that there is a connection between cellular automata and the N-soliton problem, which is of direct interest in nonlinear optics as well as other fields.[13] By adopting a generalized cellular automaton for our simulation environment, we must introduce only one pertinent restriction: namely that simulations are always pinned to a predefined lattice.[14]

II. ABSTRACTION

The most important technical feature of the CASE library is its object-oriented construction. Object orientation is about hiding details inside ''black boxes'' so that they do not interfere with the natural logical structure of a problem. It is also about separating independent issues in a program in a disciplined way. Using an object-oriented philosophy together with the polymorphism allowed by C++, we are able to create models with a structure which is independent of low-level details. By low-level details we refer to both microscopic physical details of the models we simulate and low-level programming details, such as the specifics of how to draw graphics in windows. The object model is a powerful abstraction that had largely been ignored by physicists who are more used to nuts and bolts programming with languages like Fortran. We feel that this aspect of CASE should not be underestimated. Practical models of real physics in complex systems will only succeed in the future if there is a serious attempt to deal with this complexity in a logical and organized manner.

As an example of the benefits that object orientation offers, we can note the following. Normally cellular automata in two dimensions live on a square lattice. In our framework we have been careful to admit the possibility for models to employ any regular tesselating lattice. We do this by separating issues that concern the lattice structure from as much as possible of the remaining program, and vice versa. In many cases we can code physical models in a way that is independent of the underlying symmetry of the lattice by referring only to generic concepts such as nearest neighbors. The knowledge of how to locate nearest neighbors can be hidden inside a black box associated with a particular symmetry. This allows us to model crystalline structures with hexagonal and other symmetry groups simply by replacing one black box with another. CASE is not simply a two-dimensional graphics rendering package. The library is not just about making available different geometries. It is also about simplifying and rendering different physical concepts. Any useful visualization technology must be based on general, reusable abstractions that make physical understanding paramount. Our approach is based on a fundamental untangling of concepts associated with two-dimensional cellular automata.

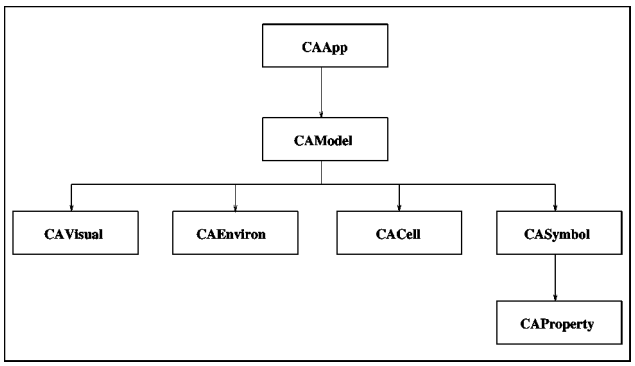


Figure 1. The CASE object hierarchy.

Figure 1 shows the way in which we have chosen to separate the issues in the CASE library. The structure of objects and the way in which they relate to one another have been designed with a practical eye: we are looking to write real computer programs, not merely theoretical constructs. The top of the hierarchy is therefore the coarsest object in our scheme: an application or complete program. The application part of our scheme is a framework in which we can open one or more simulations and run them concurrently, even in parallel. An application contains convenient control switches for starting and stopping simulations and keeping them synchronized with one another.

Beneath the application layer is the model layer. A model is a box which encapsulates everything about a given physical system, such as the rules by which physics is done. We can think of this part of the model as defining the Hamiltonian for the system (see table), since the rules

Table. Meaning of CASE objects.

CASE	Symbol	Meaning
Update( )	H	Hamiltonian
Cell object	ϕ	The field
Environ	$\mathscr{G}$	Symmetry group
Property	$T,E,\mathbf{S}$	Physical quantities

of how the system develops in time reside in this structure. This black box is itself built out of component objects that have a more general validity than one single model. For instance, ''Environment'' describes the symmetry of the lattice and the way in which one cell relates to the others. This is analogous to the symmetry group of the lattice and its boundary conditions (periodic, twisted, or isolated identification of the edges). A ''Cell'' object is a capsule that contains all of the physical variables that are required by the physical model at every site on the lattice. This is analogous to the choice of variables in the Hamiltonian, or the parametrization of the problem. Finally we have some strictly technical objects that have to do with the visual mechanisms of the computer and the definition of reusable symbols for representing the physical variables contained in cells.

This object abstraction covers a complete classification of properties associated with physical systems in a rational, formalized scheme. We can define plug-in visualization schemes for scalar, vector, or other data based on a lattice of arbitrary symmetry. We can render these properties with arbitrary shapes and colors or even with numerical values, or combinations of these objects. Visual objects might hide variables, which can be revealed by clicking on a cell to avoid visual clutter. Perhaps more importantly than the results, we are able to program the physics in a way which preserves the structure of a system at the algebraic level: cells are the basic dynamical variables, models are the Hamiltonian, and each of these can be separated without muddling issues.

The basic symbol types that have been used so far in example models are: colored blobs to represent scalar values, arrows (with variable size) to represent vector data, and the use of scaled and colored plus and minus symbols to represent the locations of positive and negative charges of varying magnitudes, particularly in connection with vortices or charges.

III. IMPLEMENTATION

The CASE simulation environment began in Oslo in 1993[15] with a simple implementation of the *XY* model programmed as a one-off application on a Unix workstation, using the X11 windowing library.[16] Since then we have developed a generalized framework for cellular or lattice based visualizations, based on state-of-the-art, freely distributable computing systems.[17,18]

Our aim has been to create a flexible system for constructing generalized, visual models sufficient for our own special needs, but which may be adopted and developed by others, without the need for expensive software packages. Indeed, all of the tools required to use our models are freely available on the Internet. Our code is written in object-oriented C++ and is designed carefully to be comprehensible and reusable by the physics community. The technical details of this framework will be published elsewhere.[17] CASE takes the form of a library of reusable functions that hide complex programming issues from the user. We envisage our user to be an intelligent scientist with an aptitude for programming, but with little patience for incomprehensible graphical window interfaces. The framework may be loosely described as follows:

- CASE supports an *N*-state cellular automaton, based on a lattice with user configurable symmetry. Each site can be edited or randomized to specify initial conditions.
- The method of visualization must be user-configurable. Simple mechanisms for adding and changing color in real-time are provided, for example.
- The size of system and choice of boundary conditions may be changed as run-time parameters.
- Resizing and zooming in and out of the lattice permits the handling of large models.
- Freeze frame and resume, with possibility of saving the state of the simulation for later continuation or analysis. Snapshots may also be sent to a printer or other device.

A. Object construction

The implementation of a model in CASE involves the definition of a model object (which in turn uses reusable cell objects), and all of its methods (functions). Here is an example model object.

```
class XYModel : public CAModel
{
public:
  XYModel( );
  XYModel(Widget new_parent);
  ~XYModel( );
  virtual void Update( );
  virtual void RightClick(double x, double y);
protected:
  virtual void Draw(int cell_index);
  virtual void Redraw( );
  virtual void AllocateCells(int new_cells);
  virtual void ReadEnvironRequester( );

  void Init_spinconfig( );
  int Monte_Carlo(int cell_index);
  double Trial_angle(XYCell *current);
  double New_energy(int cell, double trial_angle);
  double Old_energy(int cell);
  int CheckVortex(int cell_index);
  void CheckVortices( );

  CAArrowSymbol *symbol;

  double beta;
  Req temp_req;
  int iter;
private:
  static void temp_ok(Req *req, XtPointer data);
  static void temp_cancel(Req *req, XtPointer data);
  static void set_temp(Widget w, XtPointer data,
                       XtPointer garb);
};
```

The function prototypes here refer to functions that the user of CASE must supply in order to describe the nature of cells. In addition to certain required functions that control aspects of the user interface, which CASE needs in order to function (RightClick, Update, etc.), any number of other

functions may be created as is convenient for implementing the model. In the XY example, these include Init_spinconfig, CheckVortices, etc.

The most important function is the Update method. This is a formal coding of the Hamiltonian of the system. In many cases this update procedure is based upon randomized statistical processes. This is implemented using a Monte Carlo algorithm.

B. Monte Carlo update procedure

In a statistical system, close to thermodynamic equilibrium, the average state of the system is characterized by a temperature $T>0$, or equivalently by inverse temperature β. Microscopically, the state is defined by the condition of every spin in the simulation, but this is not a static state: we must allow for fluctuations. Since it is impractical to account for every degree of freedom that gives rise to such fluctuations, one adopts a heuristic algorithm for including them. In our case, this is known as the Metropolis algorithm, which is a variant of so-called Monte Carlo methods.

A Monte Carlo method is a way of using random numbers to update the state of the system (the name Monte Carlo evokes images of gambling). Instead of iteratively parsing the cellular grid and updating the state of each cell deterministically, one chooses spins in the grid according to some rule and applies a rule that only updates spins with a certain probability, characterized by the Boltzmann factor. The algorithm is as follows:

- Select a spin;
- Pick an angle by which the spin might flip at random;
- Compute the energy change associated with the random flip ΔE;
- Calculate the probability of transition $W=\exp(-\beta\Delta E)$;
- Calculate a random number x between zero and unit;
- If $x<W$, update the spin, otherwise leave it alone.

In a real system, fluctuations cause spins to flip at random. In this sense the Monte Carlo algorithm may be thought of as a true time-dependent simulation of a physical process approaching equilibrium. Note, however, that there is no immediate connection between the time elapsed in the simulation and ''physical time.'' The true physical time is modeled most realistically if we update the sites at random, but time is saved if we go through the grid in an orderly fashion. Although one would not expect any major problems to arise from this orderly parsing of the grid, it makes the identification of the physical time a more heuristic than precise procedure. In the limit of large times, the ergodicity of the system should guarantee that the specific method of updating would not play a role in the final outcome. In our example implementation of the XY model, we choose random updates to best simulate a physical system.

C. X Windows

As a graphical interface, it is natural to choose MIT's X Windows system, which is already a de facto standard in the Unix world and has been ported for use under Windows NT. The very latest X11 release is available, ready compiled for many operating systems including Solaris 2.6, Linux, and FreeBSD operating systems for the PC. The X Windows system is itself an object-oriented environment. Each window or graphical object is an ''object'' in the sense of the previous section. Sub-objects inherit the properties of their parents or can override them.

The CASE library builds on the X11 libraries and extends them. Functionality is added and none is removed. Access to low-level routines is still possible, although scarcely desirable.

A graphical user interface works in a way fundamentally different from a traditional question–answer driven interactive program. The user is presented with an image that consists of many objects which may be clicked upon or moved with the mouse pointer and is encouraged to click on any of these at any time. In this scenario, it is not the program that steers the user, by prompting questions and receiving answers, but the user who steers the program. The program must then respond to the random wishes of the user's clicks by *reacting* with an appropriate action. A system like the one described above is called an *event-driven* program and is handled by a so-called *event loop* or *reactor*. Because the user might click on a graphical object at any time, a program must be ready to receive input at any time and service that input quickly. This is achieved by centering all such interactive programs around a user interface. Each time the user clicks on a graphical object (a button or menu item), a hardware signal is sent from the mouse to the system's input/output (I/O) port. This signal is intercepted by the X Windows system and the coordinates of the mouse pointer are used to decode which object was clicked on. The X Windows system then generates a *software signal* or *event message* which is placed in a queue for processing. It is the job of an X program to process this event queue fast enough so that the events do not build up out of control.

```
while (XGetNextEvent( ))
{
case ButtonPress:
                    callback( );
other events
}
```

The X Windows system provides a programming interface to these software messages by allowing graphical objects to receive various classes of message. For each class of message, on each instance of an object (such as a button or menu), the user supplies a function which will be called by X when that message is received. An additional level of abstraction in this reactor-based system allows the event loop to be hidden in a library function. This abstraction is the *callback*.

A *callback* is a function which is called by the X Windows system each time a graphical object receives a message. When setting up a graphical user interface, such a callback function is *registered* with a graphical object (menu or button etc.). The system is therefore told what function it should call when such a message is received, and what parameters it should send to that function. Callback functions should return quickly so that the event loop does not fill up with events. By registering these functions, one builds up a custom software description of how an X

application should behave to different external stimuli. In addition to registering callbacks, user programs can register other functions which are to be performed at regular intervals or times, such as functions to update the display, or to increment a timer counter.

CASE takes care of all of these low-level details of event driven processing by providing additional abstractions with considerably simpler user interfaces than those provided in the raw X11 library. We believe it will always be easier to use the CASE library for cellular simulations rather than direct use of the X11 library.

D. Threads

In the present model we have been able to get away with a rather primitive algorithm for scheduling updates of the window graphics, based on the X timeout mechanism. This method has several failings: if the calculations are very time-consuming, then the normal X event loop becomes neglected. This means that the responsiveness of windows and button clicks will degrade and becomes unacceptably slow. This is likely to be a problem in more ambitious simulations. The only way to avoid this problem is to separate the updating of the window from the business of processing X11 messages. The most natural way to do this is to use threads. By making a simulation multithreaded, we can organize most efficiently the time spent on each task without having to break up the updating algorithm, in an unaesthetic fashion. This is also a natural object-oriented solution to the problem.

CASE currently uses release 6 of X11 and will continue to build on this and later releases. X11R6 contains a thread-compatible library with which we shall experiment in the future. One possibility is to use POSIX pthreads, but so far only a few operating systems have implemented pthreads in an acceptable way. We shall most likely try to encapsulate the threading abstraction in a suitable class to hide the chiefly technical difficulties.

IV. EXAMPLE VISUALIZATIONS

We illustrate the usefulness of our simulator with examples based on two-dimensional spin films and a diffusion model. We hope that these model simulations will be of widespread interest to physicists and students alike and lead to many helpful insights into the nature of microscopic systems both in classical, semiclassical, and quantum-mechanical descriptions. In the first two cases we focus on what can be learned from the visualization of physical systems, while in the final example we demonstrate the benefits of our abstraction policy.

A. The two-dimensional *XY* model

One of the simplest but nonetheless interesting models in two dimensions is the so-called XY model for spin systems. Considerable efforts have been expended in order to explore the properties of the two-dimensional classical XY model (for a review, see Ref. 19). There exists a set of excitations that takes this system from its low-temperature phase described by the spin-wave approximation to a simple high-temperature disordered phase. These excitations were identified by Kosterlitz and Thouless[20] to be *vortices*, which at low temperatures occur in tightly bound pairs that unbind at a critical temperature.

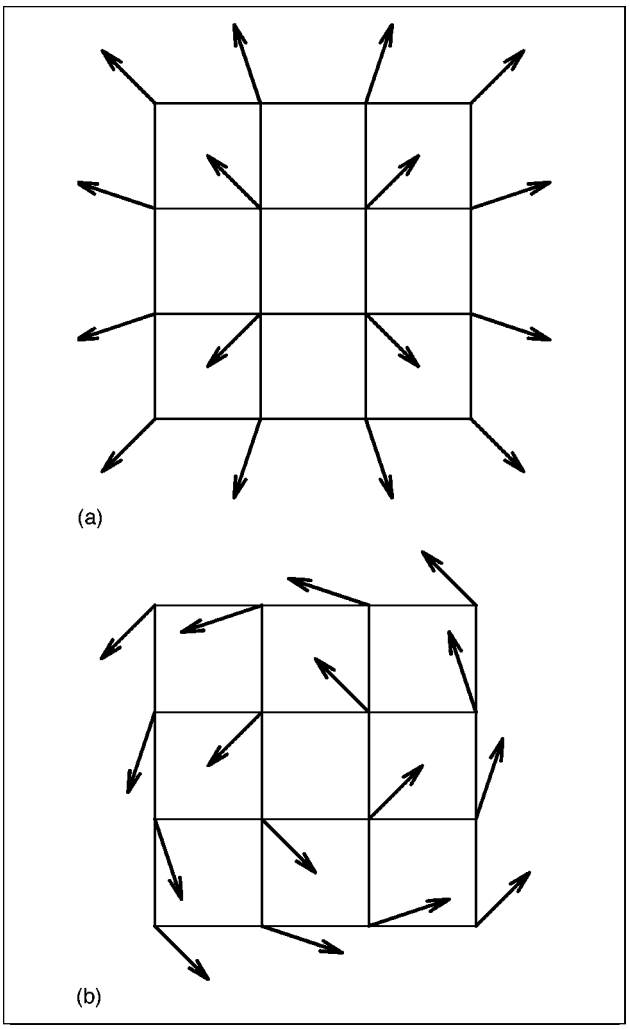
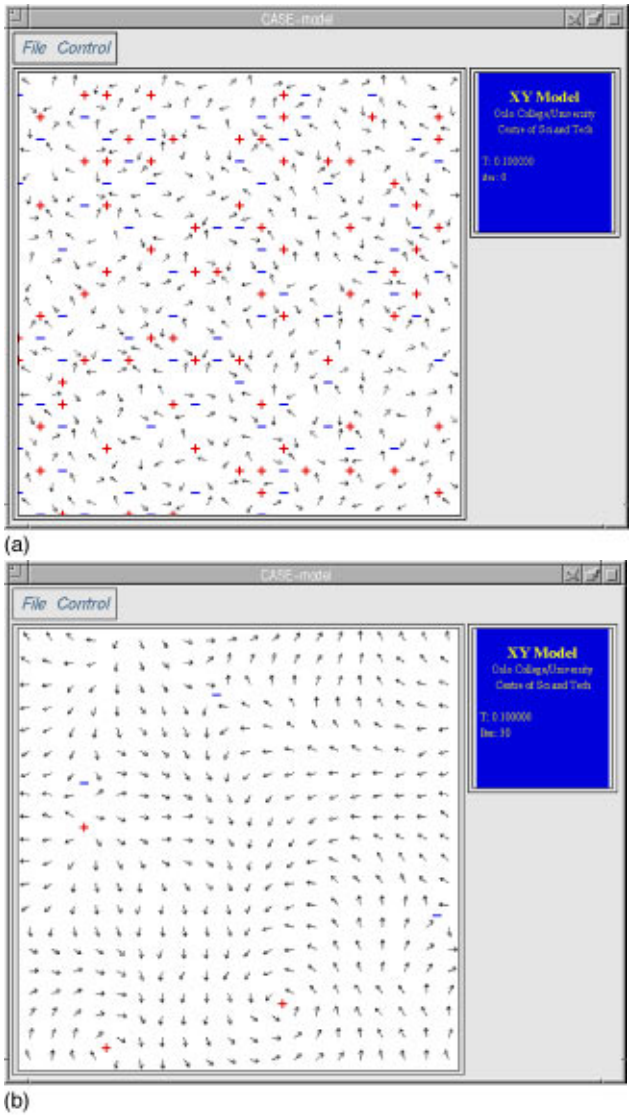


Figure 2. Vortices in the XY model: (a) All spins point in the direction away from the vortex core. (b) The spins have all been rotated an angle $\pi/2$. This whirl-shape is a characteristic of a vortex.

The XY model has attracted much interest in recent years due to its ability to describe the vortices of high-T_c superconductors. The dynamics of these vortices are very important to understand, because the existence of free vortices can destroy the zero resistance of the high-T_c superconductors. The interest in the XY model has also been motivated by the fact that it can be viewed as a microscopic model of a neutral two-dimensional superfluid[21] and thus describe the vortex excitations of ^{4}He films. Furthermore, it is closely related to the two-dimensional Coulomb gas.[22]

Due to the complexity of the model, analytic solutions are hard to find; one therefore resorts to Monte Carlo simulations to obtain estimates of thermal expectation values.[23–25]

The two-dimensional XY model is a model of classical spins $\mathbf{S}$ located at each lattice site of a two-dimensional square lattice. The name XY stems from the fact that the



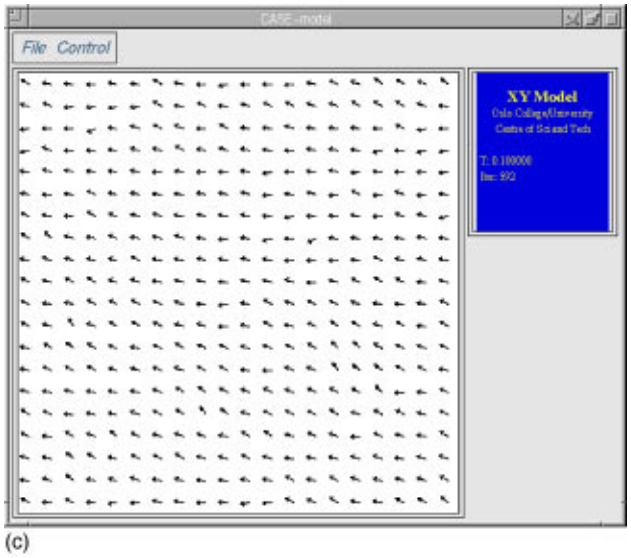


Figure 3. This figure shows the route into equilibrium at $T=0.1$ for the XY model.

spins are constrained to rotate in a plane, and the spins are classical in the way that they can point in any direction in this plane. The Hamiltonian is given by

$$H=-J\sum_{\langle \mathbf{x},\mathbf{x}'\rangle}\mathbf{S}_{\mathbf{x}}\cdot\mathbf{S}_{\mathbf{x}'}\overset{|\mathbf{S}|=1}{=}-J\sum_{\langle \mathbf{x},\mathbf{x}'\rangle}\cos(\phi_{\mathbf{x}}-\phi_{\mathbf{x}'}), \quad (1)$$

where $\langle \mathbf{x},\mathbf{x}'\rangle$ means that the sum is restricted to nearest neighbors, and J is a positive coupling constant. Here $\phi_{\mathbf{x}}$ means the angle $\mathbf{S}_{\mathbf{x}}$ makes with an arbitrary, fixed axis. An illustration of vortices on such a spin lattice is shown in Fig. 2.

According to Kosterlitz and Thouless[20] the vortices exist only in tightly bound pairs with opposite vorticity at low temperatures. At some critical temperature, T_c, the first vortex pair unbinds and the vortices are free to move to the surface of the lattice under the influence of an arbitrarily weak, external magnetic field. The phase transition is therefore called the vortex-unbinding transition, and it is characterized by this abrupt change in the response to a magnetic field.

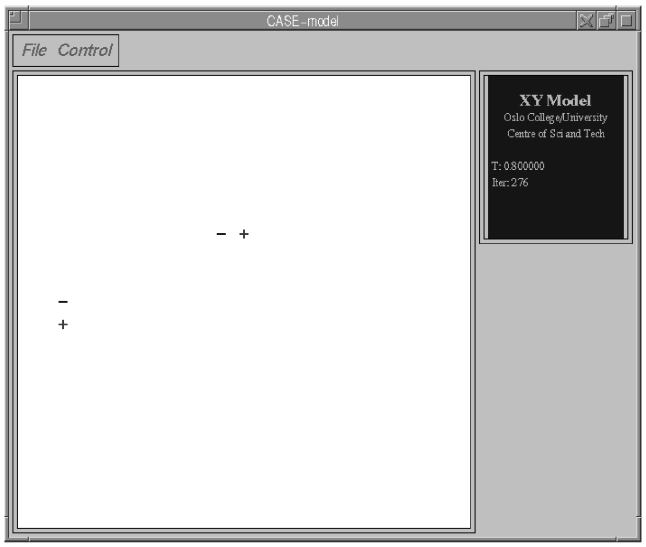
In the CASE setup, this Hamiltonian is coded into a module called XYModel, which makes reference to separate cell objects containing angle data. The statistical mechanics of the model is contained in the partition function

$$Z=\int_{-\pi}^{\pi}\prod_{\mathbf{x}}d\phi_{\mathbf{x}}\,e^{\frac{J}{kT}\sum_{\langle \mathbf{x},\mathbf{x}'\rangle}\cos(\phi_{\mathbf{x}}-\phi_{\mathbf{x}'})}. \quad (2)$$

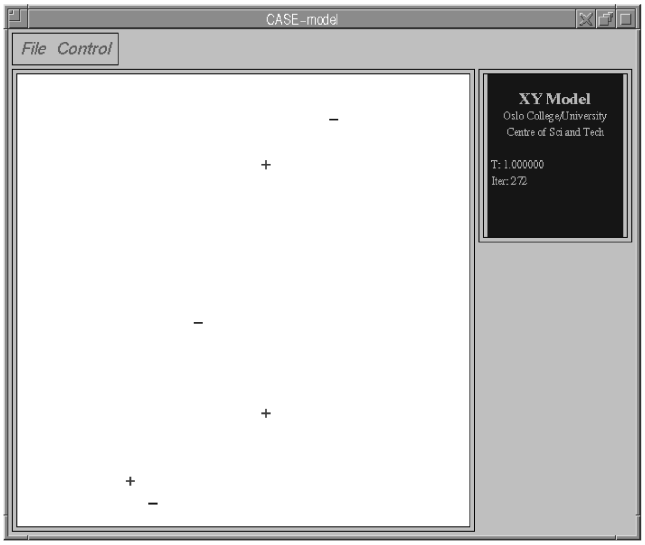
Any thermodynamic quantity of interest can be calculated through the partition function, or in our case from the model abstraction.

In order to focus on the thermal behavior of the XY model we use a Monte Carlo simulation, with a standard local Metropolis[26] updating scheme, which chooses new trial configurations by a random change in the direction of one single spin at the time. The real-time simulations reveal interesting information about the system's behavior in the nonequilibrium phase of the Monte Carlo simulations.

Figure 3 shows snapshots of the terminal screen during a low-temperature Monte Carlo simulation. The system



(a)



(b)

Figure 4. Equilibrium snapshots at T=0.8 and T=1.0. The vortices are seen to exist mostly in bound pairs. An example of vortex unbinding is shown in (b).

is seen to start out from an initial, random configuration. Soon the spin configuration is dominated by the vortices, and they start to annihilate by drifting into one another. Figure 3(b) shows a configuration, close to equilibrium, which contains a few persisting vortices. Since we use a local updating algorithm, it is reasonable that we will need a considerable number of Monte Carlo steps to take the system out of these local energy minimum configurations into the equilibrium, global energy minimum. In the end all of the spins will have to be aligned at these low temperatures as in Fig. 3(c).

We were also able to visualize the Kosterlitz–Thouless prediction that the model has a vortex-unbinding transition. It appeared that the vortices exist only in tightly bound pairs below the critical temperature and we were able to detect free vortices above the critical temperature.

Figure 4 shows some equilibrium snapshots above and below T_c. The spins are omitted here in order to emphasize the vortex behavior. In Fig. 4(a) we display a typical configuration below the critical temperature. Vortex pairs are spontaneously created and annihilated in pairs, and they always exist in pairs. For the case of a temperature above the critical temperature a different behavior is observed, as shown in Fig. 4(b). The vortices are also here created and annihilated in pairs, but here we have the possibility of a vortex pair unbinding. This event is very rare just above T_c, but it is more common at higher temperatures. The effect seems to be caused by other vortex pairs being created in between the original pair. These other pairs are then polarized and thus screen off the interaction between the original pair, thereby making it easier for the original pair to unbind.

B. The two-dimensional *XY* model in a magnetic field

When including the electromagnetic vector potential in the XY model it can be viewed as an approximate microscopic model for superconducting films as well as for two-dimensional arrays of weakly coupled Josephson junctions.[27]

It was shown in a classic article by Abrikosov[28] that the flux lines in a type-II superconductor will form a hexagonal lattice and this has also been seen experimentally.[29] The dynamics of such a three-dimensional flux line or vortex lattice has been of considerable interest after the discovery of high-temperature superconductors.[30] Based on experimental[31] and theoretical[32–34] results it was proposed that the flux-line lattice melts into a vortex liquid over large parts of the vortex phase.

A widely used microscopic model[35–37] for describing the statistical mechanics of the flux line lattice is the XY model in a magnetic field with a Hamiltonian given by

$$H=-\sum_{ij}\cos(\phi_i-\phi_j-A_{ij}), \tag{3}$$

where ϕ_i is the phase of the superconducting wave function ψ[38] at site i, the sum is over nearest neighbor sites and

$$A_{ij}=\frac{2e}{\hbar c}\int_i^j \mathbf{A}\cdot d\mathbf{l} \tag{4}$$

is the integral of the vector potential from site i to site j. This effective model can be derived by discretizing the Landau–Ginzburg free energy functional under the assumption of a spatial uniform magnetic induction and constant $|\psi|$ outside the normal vortex cores, the London approximation.[39] The flux lines are three-dimensional objects, but in some cases it turns out to be a good approximation to treat them as two-dimensional, i.e., as straight lines in the z direction. A superconducting thin film is one case and another is high-T_c superconductors that consist of weakly coupled layers, and so for some range of parameters may display effectively two-dimensional behavior.[34,40]

We consider a quadratic lattice with lattice constant a, which serves as a measure of the coherence length ξ. Each flux line carries a flux equal the flux quantum $\Phi_0=hc/2e$ and the average density of field-induced vortices is thus f

$=Ba^2/\Phi_0$. Since the total flux through a plaquette of the lattice equals $\oint \mathbf{A}\cdot d\mathbf{l}$, the sum over A_{ij} around such a square must obey the constraint

$$A_{ij}+A_{jk}+A_{kl}+A_{li}=2\pi f. \tag{5}$$

The localization of the vortices are as in the standard XY model determined by calculating the total change of the phase ϕ around a plaquette: $\Sigma_{\square}(\phi_i-\phi_j)=2\pi n$, where n is the integer vorticity. Neither the phase ϕ nor the vector potential $\mathbf{A}$ are directly observable quantities, but choosing a particular gauge fixes $\mathbf{A}$ and thereby ϕ. However, it is convenient[36] to treat the model in a gauge invariant manner, and we will follow this approach. The superconducting current density is a physical observable and can be expressed as

$$\mathbf{J}=|\psi(\mathbf{x})|^2\frac{e\hbar}{m}\left(\nabla\phi-\frac{2e}{\hbar c}\mathbf{A}\right). \tag{6}$$

Hence the factor $\alpha_{ij}=\phi_i-\phi_j-A_{ij}$ of the Hamiltonian is gauge invariant. Throughout the simulations no reference is ever made to a specific vector potential, nor to a specific phase ϕ. The Monte Carlo moves are instead performed on the gauge-invariant phase α, which is the single dynamic variable of the system. As stated above, the vorticity depends on the phase difference of ϕ only. Nevertheless it is possible to calculate the vorticity from the gauge invariant phase α since

$$\sum_{\square}(\phi_i-\phi_j-A_{ij})=2\pi(n-f), \tag{7}$$

where the value of $\alpha_{ij}=\phi_i-\phi_j-A_{ij}$ is restricted to the interval $(\pi,-\pi)$.

One of the goals of Monte Carlo simulations of vortex lattices is to the determine the phase diagram (see for instance Fig. 1 in Ref. 41) including the pinned solid Abrikosov phase and a liquid phase as well as the depinned floating solid phase. For the latter the hexagonal lattice is intact, but it is not pinned and able to move as a whole. In the simulations this corresponds to a phase where the hexagonal lattice floats on the underlying numerical lattice. For such investigations it is very useful to be able to see how the actual configurations change during the simulations, for different temperatures and concentrations of vortices.

Here we will concentrate on the formation of a hexagonal Abrikosov lattice generated when starting from high temperature and relaxing at a sufficiently low temperature. For studies of the phase diagram it is essential that a hexagonal lattice can be formed at low temperatures without frustration. In actual simulations the size of the lattices is restricted by practical concerns and in order to approximate a large system better, periodic boundary conditions are imposed. If the appropriate size of the underlying numerical lattice is not carefully chosen, this will lead to frustration of a hexagonal lattice. Since we use a quadratic numerical lattice, a perfect hexagonal lattice cannot be generated, and also a proper vortex concentration f must be chosen. The dilute case, $f\to 0$, can equivalently be viewed as the continuum limit, in which the lattice spacing a decreases to zero for a fixed areal density of vortices. So for addressing the problem of vortex lattice melting in a uniform superconducting film, the vortex concentration f should be small, probably less than 1/30 in order to see all three phases of the phase diagram. The ground-state configurations are quadratic for large concentrations,[42] even for $f=1/25$,[43] and hence f must be smaller in order to approach the continuum limit.

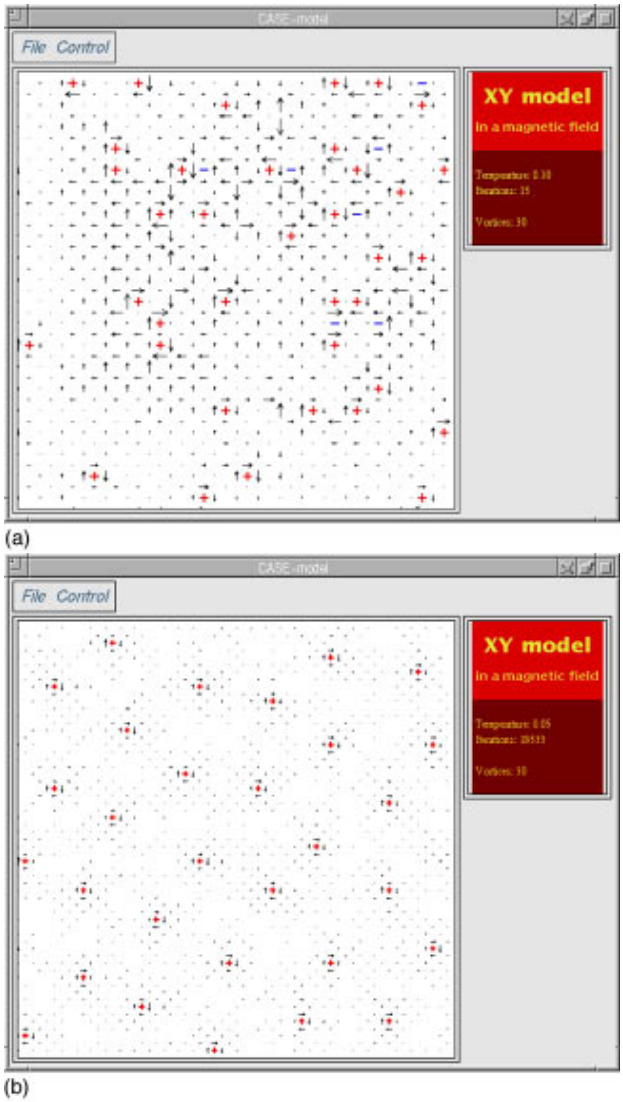
These considerations are taken into account when choosing $f=1/30$ on a lattice of 30×30 lattice sites.[44] For these parameters an almost perfect hexagonal lattice may be formed without frustration due to periodic boundary conditions. As an initial state we choose the high-temperature limit, a random configuration of vortices. After the initial configuration has been loaded, sites are chosen at random and probed by the Metropolis method. During the first few iterations the vortex configuration changes rapidly and occasionally vortex–antivortex pairs are created, as seen in Fig. 5(a). After a large number of iterations an almost hexagonal lattice is formed, as seen in Fig. 5(b). There are some fluctuations around this configuration because of the finite temperature, but the hexagonal lattice is pinned to the underlying numerical lattice and thus not moving. According to Ref. 44 one would expect the vortex lattice to melt into a vortex liquid when increasing the temperature above $T=0.045$. At this temperature both the helicity modulus, a measure of long-range phase coherence, and the sixfold orientational order parameter drops rapidly to zero, which means that depinning and melting occurs at the same temperature.

However, if the vortex concentration is lowered to $f=1/56$ the helicity modulus drops at $T_p=0.03$ and the orientational order parameter at $T_m=0.05$.[44] This means that the system at T_p enters the floating solid state where the hexagonal vortex lattice is unpinned and moves as a whole independent of the numerical square lattice. The appearance of such a phase signals the onset of the continuum limit, since the pinned phase is an artifact of the lattice model. At T_m the hexagonal lattice melts and the system finally enters the vortex liquid phase. A vortex concentration f of less than 1/30 is therefore necessary in order to study the true melting of the flux lattice.

The gauge invariant phases α are also visualized in Fig. 5. The arrows show the positive direction of the phase (current) and their size is proportional to the magnitude of the phases. Close to the vortex core there are strong currents due to the rapid change of the superconducting phase ϕ. In between the vortices the current around a plaquette is small, but nonzero. Here there is no contribution from the superconducting phase, but there is a small net current in the opposite direction due to the constant magnetic field.

C. Diffusion into a lattice

In the previous examples the models were based exclusively on a square lattice in order to simplify special highlighting algorithms. In this section we wish to present a more generic automaton that demonstrates the adaptability of CASE's object-oriented construction. We use a toy model for classical Brownian motion. The algorithm is intended to represent diffusion of particles into the surface plane of a crystalline lattice from outside. The plane being



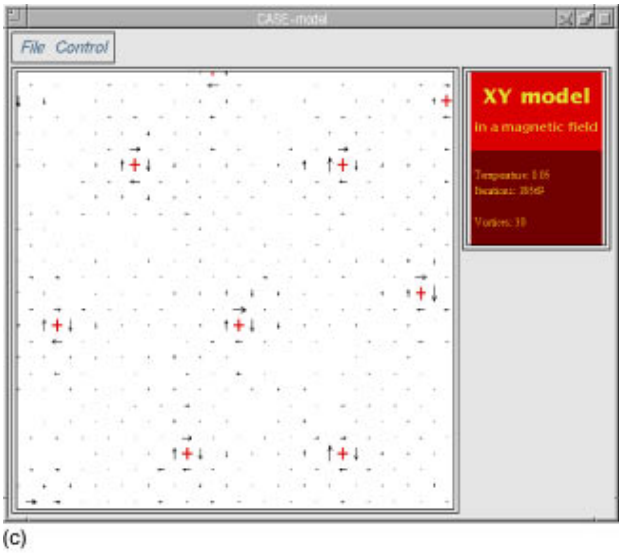
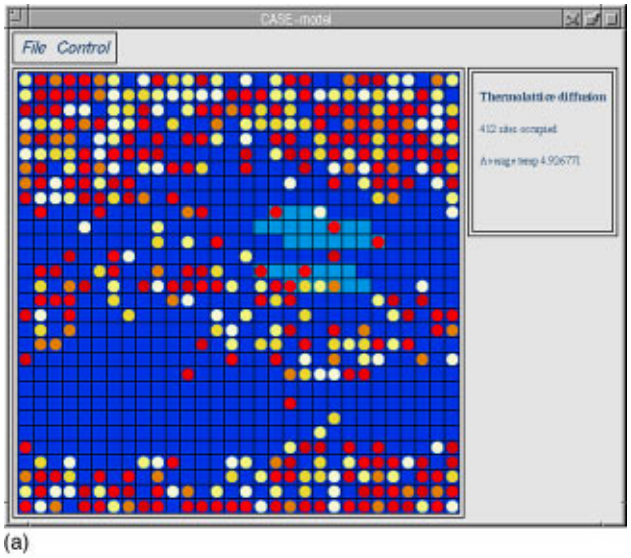


Figure 5. Snapshots of the XY model in a magnetic field. (a) shows an initial disordered state of the system. (b) shows the state of the system after a large number of iterations, where an almost hexagonal lattice of the vortices is formed. (c) An enlarged portion of the figure in (b) obtained by zooming in on the lower right hand corner with the middle mouse button, a standard function in the CASE library.

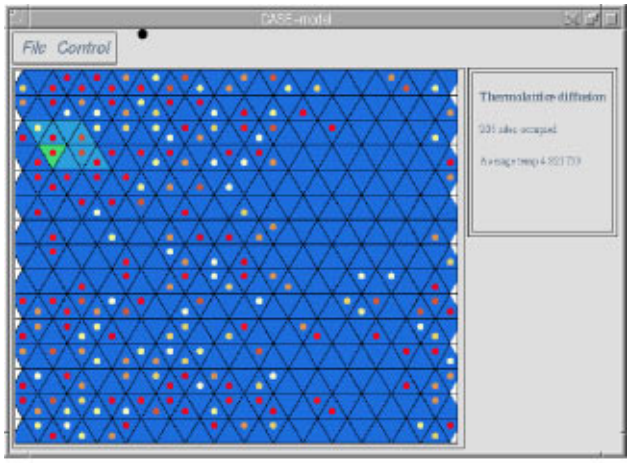
visualized should therefore be thought of as a surface layer separating a three-dimensional region of solid crystal from a corresponding region of empty space filled with gaseous particles. When particles penetrate the lattice from the gaseous region they appear spontaneously in the model at a random site. When they leave the lattice by further penetration into the bulk or by escaping back into the gaseous region, they disappear from the model. This is used as a simple toy model that may be developed into a model for the way in which atomic hydrogen penetrates a palladium lattice.

The geometry of the lattice is completely hidden from the model algorithms and we can therefore run several simulations with different lattice structures side by side and compare them without having to rewrite any code whatsoever. It is not necessary to know the specifics about the background geometry: simple abstractions such as ''nearest neighbor'' suffice. To determine possible destinations for particle hopping, we need only to obtain a list of nearest neighbors from the lattice-specific object. Figure 6(a) illustrates the diffusion model implemented on a rectangular lattice, while Fig. 6(b) shows the same model on a triangular lattice. The difference between these images is a single switch: no special code is required in the model to make this change. Figure 6(c) shows a larger simulation with some background color highlighting.

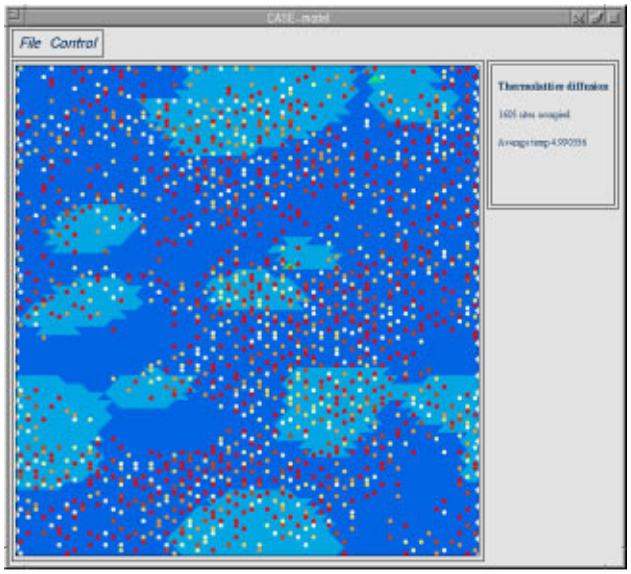
The dynamics of the model are derived from simple energy considerations. To make the model physically reasonable, we must have an exchange of energy and a Monte Carlo based algorithm for randomly perturbing the system. For this simple demonstration we choose to illustrate an interaction between the thermodynamical properties of the lattice and the arrival and disappearance of particles. When a new particle enters the system, it contributes an amount of energy that warms up the lattice locally. This energy input is a combination of kinetic energy and energy of dissociation of a chemical bond as molecular hydrogen becomes atomized. The values of these energies are set to constant values in the model object. The extra heat contributed to the lattice is visualized by a change in the background color from blue (cold) to green (warm). The change occurs locally, but on each iteration of the model, an averaging procedure is used to conduct away local hot spots and maintain thermal equilibrium. The color of the particle



(a)



(b)



(c)

Figure 6. Snapshots of the thermodiffusion simulations: (a) shows the model implemented on a square lattice, (b) shows the model on a triangular lattice, and (c) illustrates the behavior of the color highlighting in the triangular case after some time.

blobs represents the energy of the particles and uses a different color scale for clarity. The energy of the particles is chosen randomly for the sake of illustrating a variety of colors.

Particles are selected at random and can hop over to nearest neighbor sites with a certain probability; this introduces a Brownian motion into the system. The probability for hopping depends on an energy barrier or work function which we have chosen to scale according to the square root of the average temperature between nearest neighbor sites. This crudely simulates a classical harmonic barrier of height x where $\frac{1}{2}kx^2 = k_B T$. Thus increased thermal activity of the lattice makes diffusion harder at constant particle energy. The following code snippet illustrates a simple Brownian motion algorithm using the lattice-independent mechanisms.

```
// Find the nearest neighbors at 1-st level
// by querying lattice
grid->QueryNeighbours(i, 1, neighbors);
for (i = 0; i < neighbors.GetSize( ); i++)
  {
  if (neighbors.array[i] == environ->NoCell)
      {
      continue; // Edge effect
      }
neigh = (ThermoCell *) cell[neighbors.array[i]];
if (neigh->Alive( ))
  {
  continue;
  }
// Get energy for required processes
beta = Tscale / (current->T + neigh->T);
transition_prob =
  exp(-beta*Work(neigh,current,cell_index,i));
// Note 'least action'
if (transition_prob > highest)
      {
      highest = transition_prob;
      favorite = neighbors.array[i];
      }
  }
// If we have no free neighbor sites
if (highest == 0.0)
  {
```

```
  return;
  }
// Move if the roulette wheel favours the wicked ...
if (highest >= drand48( ))
  { neigh = (ThermoCell *) cell[favourite];
  current->SetAlive(False);
  neigh->SetAlive(True);
  neigh->E = current->E;
  Draw(cell_index);
  Draw(favourite);
  }
```

Notice how we avoid referring to the specific details of the lattice by only using the lists of neighbors that are returned by the QueryNeighbours function.

As one would expect in a thermalizable system, equilibrium is achievable here after a number of iterations. The escape of a particle from the lattice is possible if it can borrow enough thermal energy from the local lattice site, with a certain probability, which may be defined in the model. This results in a local cooling of the lattice. The higher the temperature of the lattice, the greater the probability of escape. In Fig. 6(c) one sees lighter areas of locally higher temperature. After a certain time one sees the equilibration in relation to the environmental parameters in the model. The number of particles in the lattice stabilizes: as the temperature increases and evens out, the likelihood of particles escaping increases.

V. CONCLUSIONS

We have presented a framework for simultaneously visualizing and computing numerical quantities from cellular or lattice simulations, using a scheme of abstractions that faithfully separates independent issues. Our framework is based on an object-oriented analysis of the key elements of physical systems on a lattice. As an example we present the XY model for electron spins at finite temperature and show the approach to thermal equilibrium in both the vortex phase and the ferromagnetic phase. Using the visual simulation it is possible to see the Kosterlitz–Thouless phase transition in this model: tightly bound pairs of vortices below a critical temperature, and vortex unbinding above the critical temperature. We are further able to visualize the gauge invariant currents surrounding quantum mechanical vortices in a magnetic field. A toy model for diffusion illustrates the lattice independence possible in some cases.

Our computer programs are suitable for Unix workstations running release 6 of the X11 window system (Linux or FreeBSD for instance). More information about how to collect and adapt these simulations may be found at the WWW site http://www.iu.hioslo.no/~cell. We may also be contacted by e-mail at case@iu.hioslo.no.

ACKNOWLEDGMENTS

The authors would like to thank F. Ravndal for stimulating this visualization project and A. Sudbø for introducing us to the interesting idea of visualizing the XY model in external magnetic field. We also very much like to draw attention to the essential contributions made by J. Mikkelsen and T. E. Sevaldrud in the design and implementation of the CASE library as part of their dissertation in computer science at Oslo College.

REFERENCES

1. See for collected papers, *Fractional Statistics and Anyon Superconductivity*, edited by F. Wilczek (World Scientific, Singapore, 1990).
2. For a review, see: A. Karlhede, S. A. Kivelson, and S. L. Sondhi, ''The Quantum Hall effect: The Article,'' Lectures presented at the 9th Jerusalem Winter School on Theoretical Physics (1992).
3. R. Feynman, ''Simulating physics with computers,'' Int. J. Theor. Phys. **21**, 467 (1982).
4. B. P. Zeigler, ''Discrete event models for cell space simulation,'' Int. J. Theor. Phys. **21**, 573 (1982).
5. M. Minksky, ''Cellular Vacuum,'' Int. J. Theor. Phys. **21**, 537 (1982).
6. K. Zuse, ''The computing universe,'' Int. J. Theor. Phys. **21**, 589 (1982).
7. A. Schadscheider and M. Schreckenberg, ''Cellular automata for traffic flow,'' cond-mat/9511037, LANL database.
8. E. Dan Dahlberg and J. G. Zhu, ''Micromagnetic Microscopy and Modelling,'' Phys. Today **48**(4), 34 (1995).
9. The Argus project, http://www.geog.le.ac.uk/argus/About/aboutargus.html.
10. The Virtual Laboratory, http://www.tp.umu.se/TIPTOP/VLAB/about.html.
11. B. E. Kane, L. N. Pfeiffer, and K. W. West, Phys. Rev. B **46**, 7264 (1992); M. Johnson, Appl. Phys. Lett. **63**, 1435 (1993); M. Johnson, Phys. Rev. Lett. **70**, 2142 (1993); M. Burgess and M. Carrington, Phys. Rev. B **52**, 5052 (1995).
12. S. Wolfram, *Cellular Automata and Complexity* (Addison Wesley, Reading, MA, 1994).
13. T. Tokihiro, D. Takahashi, J. Matsukidaira, and J. Satsuma, Phys. Rev. Lett. **76**, 3247 (1996).
14. Note that our code is constructed in such as way as to make this restriction removable at a later time.
15. Our XY simulation was first demonstrated publicly at the 1994 workshop on anyon physics, Norway.
16. The X Window system (O'Reilly and Associates), Vols. 0–10.
17. J. A. Mikkelsen and T. E. Sevaldrud, Ing. dissertation, Oslo College, Centre of Science and Technology, 1996.
18. There are several available tools for mathematical modeling: Mathematica, Maple, IDL, etc., but these are commercial programs designed for a much wider purpose than we require, and are therefore inefficient for numerical iteration combined with graphics. Also the new language Java turns out to be too slow and not sufficiently mature to be useful for our purposes; moreover, they cannot handle the abstractions that CASE uses for modeling a physical environment.
19. M. Suzuki, in *Physics of Low-Dimensional Systems*, Proceedings of Koyoto Summer Institute, edited by Y. Nagaoka and S. Hikami (Prog. Theor. Phys., Publication Office, Kyoto, 1979), p. 39.
20. J. M. Kosterlitz and D. J. Thouless, J. Phys. C **6**, 1181 (1973).
21. T. Matsubara and H. Matsuba, Prog. Theor. Phys. **16**, 416 (1956); **17**, 19 (1957).
22. P. Minnhagen, Rev. Mod. Phys. **59**, 1001 (1987).
23. J. Tobochnik and G. V. Chester, Phys. Rev. B **20**, 3761 (1979).
24. N. Schultka and E. Manousakis, Phys. Rev. B **49**, 12071 (1994).
25. *The Monte Carlo Method in Condensed Matter Physics*, edited by K. Binder (Springer, Berlin, 1992).
26. N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller, J. Chem. Phys. **21**, 1087 (1953).
27. Y. E. Lozovik and S. G. Akapov, Solid State Commun. **35**, 693 (1980).
28. A. A. Abrikosov, Zh. Eksp. Teor. Fiz. **32**, 1442 (1957).
29. U. Essmann and H. Träuble, Phys. Lett. **24A**, 526 (1967).
30. For reviews, see G. Blatter, M. V. Feigelman, V. B. Geshkenbein, A. I. Larkin, and V. M. Vinokur, Rev. Mod. Phys. **66**, 1125 (1994); E. H. Brandt, Rep. Prog. Phys. **58**, 1465 (1995).
31. P. L. Gammel, L. F. Schneemeyer, J. V. Wasczczak, and D. J. Bishop, Phys. Rev. Lett. **61**, 1666 (1988).
32. D. R. Nelson, Phys. Rev. Lett. **60**, 1973 (1988).
33. A. Houghton, R. A. Pelcovits, and A. Sudbø, Phys. Rev. B **40**, 6763 (1989).
34. D. S. Fisher, M. P. A. Fisher, and D. A. Huse, Phys. Rev. B **43**, 130 (1991).

35. Y.-H. Li and S. Teitel, Phys. Rev. Lett. **66**, 3301 (1991).
36. R. E. Hetzel, A. Sudbø, and D. A. Huse, Phys. Rev. Lett. **69**, 518 (1992).
37. M. Franz and S. Teitel, Phys. Rev. B **51**, 6551 (1995).
38. M. Tinkham, *Introduction to Superconductivity* (McGraw-Hill, New York, 1975).
39. Y.-H. Li and S. Teitel, Phys. Rev. B **47**, 359 (1993).
40. L. I. Glazman and A. E. Koshelev, Phys. Rev. B **43**, 2835 (1991).
41. S. A. Hattel and J. Wheatly, Phys. Rev. B **50**, 16590 (1994).
42. S. Teitel and C. Jayaprakash, Phys. Rev. Lett. **51**, 1999 (1983).
43. Y.-H. Li and S. Teitel, Phys. Rev. B **49**, 4136 (1994).
44. S. A. Hattel and J. Wheatly, Phys. Rev. B **51**, 11951 (1995).