Interactive Visualisation of Spins and Clusters in Regular and Small-World Ising Models with CUDA on GPUs

A. Leist[1], D.P. Playne, K.A. Hawick

Computer Science, Massey University Albany, North Shore 102-904, Auckland, New Zealand

Abstract

Three-dimensional simulation models are hard to visualise for dense lattice systems, even with cutaways and fly-through techniques. We use multiple Graphics Processing Units (GPUs), CUDA and OpenGL to increase our understanding of computational simulation models such as the 2-D and 3-D Ising system with small-world link rewiring by accelerating both the simulation and visualisation into interactive time. We show how interactive model parameter updates, visual overlaying of measurements and graticules, cluster labelling and other visual highlighting cues enhance user intuition of the model's meaning and exploit the enhanced simulation speed to handle model systems large enough to explore multi-scale phenomena.

Keywords: visualisation, Ising model, CUDA, GPU, instrumentation

1. Introduction

A commonly recurring problem in scientific visualisation is to "see inside" a block of three dimensional data that is associated with a simulation model. Many physical and engineering models in materials science, fluid dynamics, chemical engineering and other application areas including medical reconstruction [1] fit this problem pattern. Volume rendering [2] is a relatively long standing problem in computer graphics [3] with a number of approaches for "seeing into volumes" [4] and for providing visual cues into volumes [5] having been explored. Approaches vary from identifying the surfaces present [6] in the interior of a data volume [7] and either colouring them or texturing [8] them accordingly.

It is often hugely useful to a scientific modeller to be able to visualise an interactively running model and experiment with different parameter combinations prior to committing computational resources to detailed statistical investigations. The visualisation often leads to some insight into a particular phenomena that bears close numerical study or might lead to a hypothesis or explanation as to a numerical observation such as a phase transition or other critical phenomena. Some data models require visualisation of complex data fields that may have vector elements [9] or complex numbers [10]. We focus solely on single-valued scalar voxel element data.

In recent times the notion of computational-steering [11] has been identified as attractive to users, as has the potential use of remote visualisation [12] to link fast graphics processing devices to high-performance compute systems to enable interactive real time simulation of large scale model systems with associated real time volume rendering [13] or visualisation [14]. To this end data-parallel computing systems such as the MASPAR [15] and other specialist hardware [16] were employed. An obvious continued goal is to incorporate on-demand high-performance simulation into fully interactive virtual reality systems [17]. While this goal is still some economic way away for desktop users, modern Graphical Processing Units (GPUs) do offer some startling performance potential for both computation [18, 19, 20] and volume visualisation [21] for interactive simulations in computational science.

Email addresses: a.leist@massey.ac.nz (A. Leist), d.p.playne@massey.ac.nz (D.P. Playne), k.a.hawick@massey.ac.nz (K.A. Hawick)

[1]Author for Correspondence, Phone: +64 9 414 0800, Fax: +64 9 441 8181

In this paper we consider some of the recent high-performance hardware capabilities of GPUs and just as importantly recent software developments that have led to programming languages for accelerator devices like NVIDIA's Compute Unified Device Architecture (CUDA) [22]. GPUs can help accelerate the numerical simulation as well as accelerate the rendering performance of the graphical system. Together, these two capabilities allow exploration of model systems that are large enough and yet can be simulated fast enough in interactive time, to spot new and interesting phenomena.

The application example we use in this paper is the well known Ising model of a computational magnet [23]. This model has been well-studied numerically in two and three dimensions and also graphically in two dimensions. We harness the power of GPUs to show how both the computations and the graphical rendering can be accelerated sufficiently so that a system of sizes up to around $4096x4096$ or $256x256x256$ individual spin cells can be simulated interactively.

Of particular interest to us are the changes that arise in systems like the Ising model when so-called small-world links are introduced. The Ising model is normally studied on a regular planar or cubic lattice. We have investigated how the pathways and distance properties of the system change when some of the nearest neighbouring bonds in the lattice are rewired to provide short cuts across the space of the system. Since the critical properties of the Ising model depend upon the length scales of spin correlations that the model system can sustain, adding small-world shortcut links provides an interesting way to shift the critical temperature in a precise way. The small world re-wiring probability p is therefore a useful quantitative parameter to adjust and this allows the Ising model system with its known parameters and properties to be used to compare with less well-studied critical systems including those with more complex small-world networks.

A spinning brick of coloured cells is not particularly unusual although we have managed to compute and render quite high resolution model systems. It does give new insights into these models to look at them at higher resolutions since the multi-length scale properties of critical phenomena do show up better when there are several logarithmic scales to examine. We believe we make a more important contribution by illustrating some ways that the block of data can be teased apart to see inside it and how various rendering techniques more commonly used in computer games, can be used to see into the three dimensional system. We also experiment with various dynamically computed metrics that can be plotted live, superposed on the visualisation and which help the user build up an intuition as to the meanings of parameters and the evolutionary processes in the model. Visualising the evolving simulated system is of great value in exploring how individual small-world links change its properties. We also show how additional graphical rendering techniques can highlight the rewired links and their effects upon the system.

These techniques would apply equally to other simulated models such as solvers of partial differential equations and systems that have well-posed temporal dynamics. In this paper we are considering the Ising model with an artificial Monte Carlo dynamical scheme to update or evolve the spins through the model phase space. The well known Metropolis [24] or Glauber [25] algorithms "hit" individual spins and simulate the Ising model magnet in contact with a heat-bath. It is however well known that these sort of model slow down due to the multiple length scale phenomena that dominate near the critical temperature. An excellent solution to speeding up the Monte Carlo sampling of such models is to use a cluster update algorithm that computes the correct thermal probabilities for updating a whole cluster or clump of spins at once. Algorithms such as those of Swendsen and Wang or Wolff [26] drastically accelerate the Monte Carlo sampling and we have successfully used Wolff's algorithm in numerical studies of the small-world Ising system. However, even in the case of the regular Ising system it is not a trivial matter to visualise and understand what is going on in the system when an algorithm like Wolff's is applied. In this paper we therefore also show how we can visualise the spin clusters and help the user build up an intuition between the cluster formation scales and measured numerical statistics.

This cluster visualisation is related to some Ising model specifics but visualising physical clusters and other artifacts is an important aid to studying structure formation in other model systems such as fluid models of turbulence, and we therefore believe these techniques are of broader interest to computational scientists.

In Section 2 we describe key features of the implementation that allow it to interactively visualise the simulation of a large system. We demonstrate how the visualisation can help in the understanding of complex systems in Section 3. We discuss the performance of our implementations in Section 4 and offer some conclusions for use of these techniques in general studies of complex systems and some areas for future work in Section 5.

2. Implementation

The visualisation of the Ising model simulation extends CUDA implementations of the Metropolis Ising algorithm [27] and Wolff cluster update algorithm. OpenGL is used for the visualisation of the simulation and rendering of additional information, such as graphs that show certain system properties and how they change over the duration of the simulation. Algorithm 1 describes the major steps performed by the system.

Algorithm 1 Pseudo-code describing the major tasks performed to generate a single frame.

Input parameters: *STEPS* simulation steps are performed before the visualisation is updated (default 1). *N* is the system size.
for $i \leftarrow 1$ to *STEPS* **do**
  do in parallel on the device: evolve simulation and collect data for the energy, magnetisation and number of spins flipped
do in parallel on the device using *N* threads: update cell colours VBO
render cells with OpenGL
render links, plots, etc. with OpenGL if required

One of the main tasks is to visualise the current spin value of every cell using colour coded points or cubes. As CUDA is used to perform the actual simulation on the GPU, the spin values reside in graphics device memory. Using CUDA to also update the colour vertex buffer object (VBO), which is used by OpenGL to draw the vertices that make up the cells in the desired colours, is therefore an obvious step. Not only does this mean the massively parallel computational power of the GPU can be exploited once more, but it also means that the spin and colour data always stays in graphics device memory and does not need to be transferred back to host memory. Algorithm 2 shows the CUDA-OpenGL interoperability as well as the actual CUDA kernel implementation used to update the cell colours in a 3D simulation.

The original simulation code was extended to set the `MASK_FLIPPED` bit described in Algorithm 2 and to obtain data which is required to generate real-time plots for the fraction of spins flipped in the previous simulation step, as well as the current energy and magnetisation. Algorithm 3 illustrates how these changes were implemented.

3. High Performance Simulation and Visualisation

This section showcases how the combination of parallel, high performance simulation and visualisation can provide important insights into both the state of the system at a particular simulation step as well as the ongoing real-time state changes. In addition to visualising the current spin of every cell graphically using different colours, a number of live-plots show how certain system properties change over time. There are currently plots for the magnetisation, energy, and fraction of spins flipped in a single simulation step—which is equal to the cluster size when using Wolff's algorithm. There are also two graphs showing the standard deviations for the energy and fraction of flipped spins.

Figure 1 illustrates two 2D Metropolis Ising simulations with fixed temperatures after 1000 simulation steps. The system on the left uses a regular lattice with periodic boundaries, while the second image shows a lattice that had a small fraction of its edges randomly rewired. Several metrics are calculated after every simulation step and visualised using plots to provide information about changes to important system properties. This is a useful tool when investigating a critical phenomena of the system.

The average degree of a cell increases from 4 on a 2D lattice to 6 on a 3D cubic lattice, where every cell is connected to the cell in front and behind of itself in addition to the cells on the right, left, above and below. Figure 2 visualises a 3D Metropolis Ising simulation with no rewiring after 1000 and after 2000 simulation steps. It also demonstrates how the cube can either be split open layer by layer or completely unfolded into a grid view to gain an insight into what is happening on its inside.

The ability to adjust simulation parameters while the simulation is running and the instant feedback provided by real-time visualisation are helpful tools when analysing a system. They can give new insights into the impact of a certain parameter or combination of parameters on the system properties and behaviour. This can speed-up the process of narrowing the parameters down to the most interesting values before extensive computational resources are committed for detailed statistical investigations. Figure 3 demonstrates this by reducing the system temperature in intervals of 250 simulation steps. The effects of these parameter changes on the system properties can clearly be seen in the changes visible on the plotted graphs.

Algorithm 2 The host function `updateColours` maps the VBO that specifies the vertex colours to OpenGL into the CUDA address space and executes the CUDA kernel `colours_kernel_3d` which updates the values according to the current spin values of the individual cells. The `MASK_FLIPPED` bit is set on spins that have been flipped since they have last been visualised and is used to highlight this recent change (see Figure 4).

```
void updateColours(cudaPitchedPtr spin, GLuint coloursVboId, int dimXLen, int dimYLen) {
  cudaGLRegisterBufferObject(coloursVboId); // register VBO to CUDA
  float4* d_colours;
  cudaGLMapBufferObject((void**)&d_colours, coloursVboId); // map VBO into CUDA namespace

  colours_kernel_3d<<<gridSize, blockSize>>>(spin, d_colours, dimXLen, dimYLen); // run kernel
  cudaThreadSynchronize(); // block until the device has completed
  cudaGLUnmapBufferObject(coloursVboId); // unmap VBO
  cudaGLUnregisterBufferObject(coloursVboId);
}

__global__
void colours_kernel_3d(cudaPitchedPtr spin, float4* colours, int dimXLen, int dimYLen) {
  const unsigned int tid = (blockIdx.y * gridDim.x + blockIdx.x) *
                           blockDim.x + threadIdx.x; // unique thread ID
  const unsigned int ix = tid % dimXLen; // the cell's x-coordinate
  const unsigned int iy = (tid / dimXLen) % dimYLen; // the cell's y-coordinate
  const unsigned int iz = tid / (dimXLen*dimYLen); // the cell's z-coordinate

  // accessing the 3-dimensional spin array
  size_t slicePitch = spin.ysize * spin.pitch;
  char* slice = (char*)spin.ptr + iz * slicePitch;
  unsigned char* row = (unsigned char*)(slice + iy * spin.pitch);
  unsigned char myValue = row[ix]; // read the current spin value including marker bits
  unsigned char mySpin = myValue & MASK_SPIN; // remove marker bits

  row[ix] = myValue & (~MASK_FLIPPED); // unset MASK_FLIPPED bit and write back to memory
  float4 colourSpin0 = myValue & MASK_FLIPPED ?
    make_float4(0.0f, 0.6f, 0.0f, 1.f) : make_float4(0.f, 1.f, 0.f, 1.f);
  float4 colourSpin1 = myValue & MASK_FLIPPED ?
    make_float4(0.35f, 0.35f, 0.35f, 1.f) : make_float4(0.f, 0.f, 0.f, 1.f);
  colours[tid] = mySpin == 0 ? colourSpin0 : colourSpin1; // write cell colour
}
```

Algorithm 3 This mix of pseudo-code and actual CUDA code describes how the original simulation was extended to count the number of spins flipped in one simulation step, the number of like-like bonds and the ratio of up vs. down spins. The latter two are required to calculate the energy and magnetisation. The results of these metrics can be visualised using real-time graphs.

```
if first thread in thread block then
    initialise shared memory counters to 0
__syncthreads(); //barrier synchronisation for all threads in the same thread block
if spin value changed then
    set the MASK_FLIPPED bit on the spin value //see Algorithm 2
    atomicAdd(&blockFlipped, 1); //increment the shared memory counter for flipped spins
c ← 0 //local counter for the like-like bonds between neighbouring cells
for all neighbouring cells do
    if equal spin values then
        c ← c + 1
atomicSub(&blockE, c); //shared memory counter for the energy
atomicAdd(&blockM, (currentSpin == 0 ? -1.0 : 1.0)); //shared memory counter for the magnetisation
__syncthreads();
if first thread in thread block then
    //one atomic operation per thread block and counter to write the value to mapped system memory
    atomicAdd(magnetisationData, blockM);
    atomicAdd(energyData, blockE);
    atomicAdd(flippedSpins, blockFlipped);
```

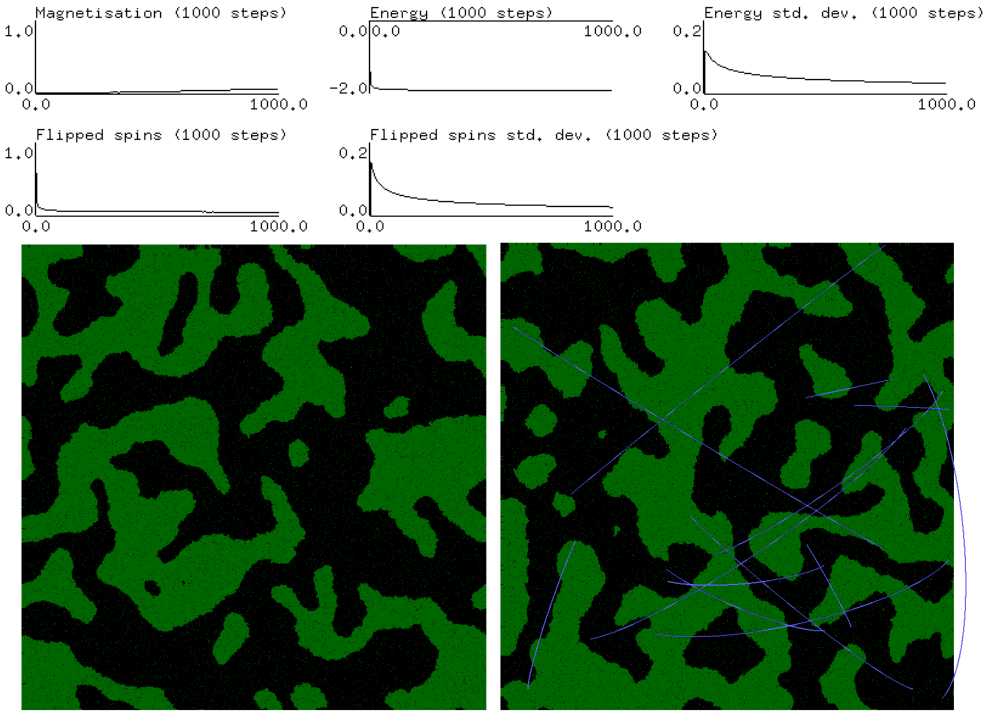

Figure 1: 2D Metropolis Ising model simulation with 2048^2 cells, rewiring probability $p = 0.0$ (left) and $p = 10^{-6}$ (right) and temperature $T = 1.8$ after 1000 simulation steps. The plots show the evolution of the the rewired simulation.

Wolff's cluster update algorithm performs particularly well near the critical temperature where the Metropolis Monte Carlo update algorithm slows down. It calculates the thermal probabilities for updating a whole cluster of cells instead of doing this for each cell individually. As shown by the plots in Figure 4, Wolff's algorithm gives very abrupt system changes after a long period of only marginal fluctuations. The images demonstrate how Wolff clusters are visualised. The cluster of cells which had there spins flipped during the previous simulation step is highlighted in a different colour that depends on the new spin value. A cluster is only highlighted until a new cluster replaces it in the next simulation step. Not only do the clusters constantly change because the boundary conditions are different every time a cluster is flipped, but Wolff clusters also have a random component that depends on the temperature. It is often a good idea to advance the simulation step by step when it gets close to a phase transition as this can happen very suddenly. The example also gives a good demonstration of how rewiring affects the simulation. If it was not for the single rewired edge in the top left image that has its endpoints highlighted by circles, the two clusters (pointed to by arrows) would be disjoined and the spins of their cells could not have been flipped together.

4. Discussion and Summary

Table 1 gives approximate performance results for the visualisation of the Metropolis Ising simulation for large systems with $N = 4096^2$ and $N = 256^3$ cells. The simulation itself, without visualisation and statistics computation, runs about 3.8 (3D) to 6.5 (2D) times faster than the single-GPU implementation. At this time no performance data is given for the Wolff cluster update algorithm, as the implementation of the simulation has not yet been fully optimised. However, while systems of up to 2048^2 or 128^3 cells provide a smoother experience with the current implementation of Wolff's algorithm, it is possible to run systems of the same size as for the Metropolis Ising simulation.

The platform used to do the performance measurements that we report, runs the Linux distribution Ubuntu 9.10 64-bit. It uses an Intel® Core™2 Quad CPU running at 2.33GHz with 8GB of DDR2-800 system memory and two NVIDIA® GeForce® GTX 260 Core 216 graphics cards, which have 896MB of global memory each on board.

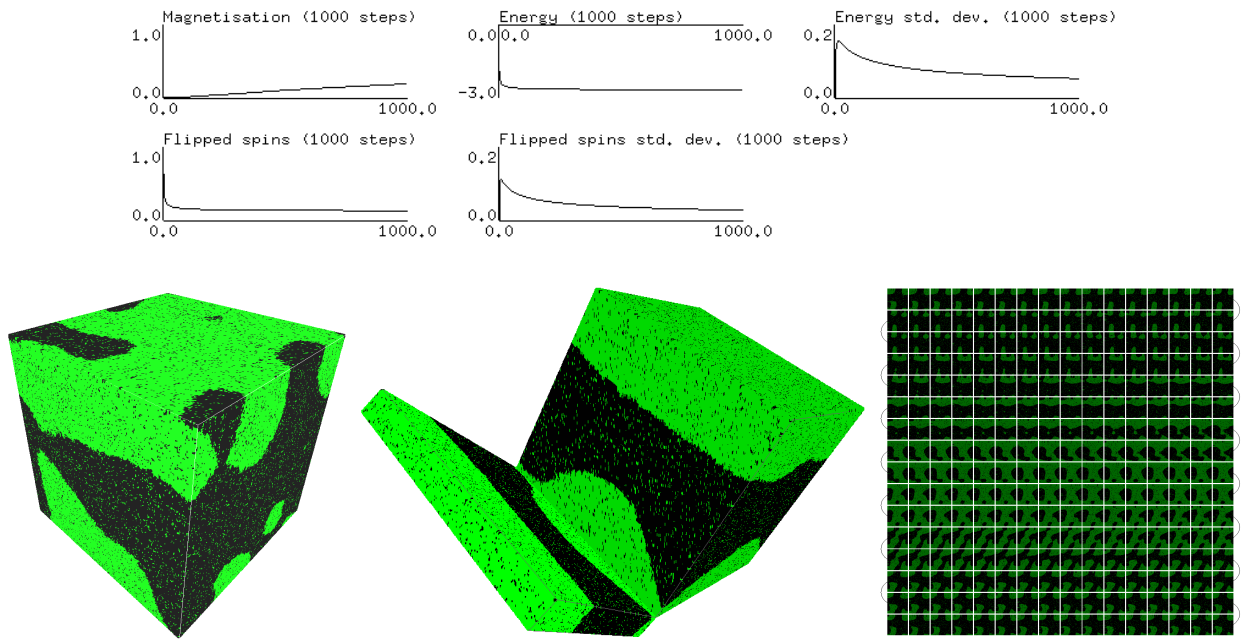

Figure 2: Regular 3D Metropolis Ising model simulation with 256^3 cells and temperature $T = 3.5$ after 1000 (left and right) and after 2000 (middle) simulation steps. The visualisation in the middle demonstrates how the cube can be split open layer by layer to give an impression of what is happening on its inside. The grid on the right arranges the z-layers in a zigzag pattern, thus providing a 2D view of the 3D system. The plots show the system behaviour over the first 1000 steps.

Metropolis Ising Performance	2D ($N = 4096^2$) - 1 GPU		2D ($N = 4096^2$) - 2 GPUs		3D ($N = 256^3$) - 1 GPU		3D ($N = 256^3$) - 2 GPUs	
	w/o stats.	show stats.	w/o stats.	show stats.	w/o stats.	show stats.	w/o stats.	show stats.
$p = 0.0$; w/o edges	10	4	33	5	11	4	23	6
$p = 10^{-5}$; w/o edges	10	4	31	5	10	4	23	6
$p = 10^{-5}$; show edges	10	4	34	5	8	4	20	6

Table 1: Approximate performance data for the Metropolis Ising simulation in simulation steps per second. This is equal to frames per second for the single-GPU implementation. The dual-GPU version uses one device for the simulation and the other one for rendering. The latter renders as many frames per second as it can, possibly skipping a simulation step if it can not keep up with the simulation. The performance was measured for 2D and 3D simulations with and without (w/o) the overhead of collecting the data and calculating the statistics for the real-time plots, and with the rendering of rewired edges enabled or disabled.

When only one of the available graphics devices is used for both the CUDA simulation and the OpenGL rendering, then it can only perform one of the two tasks at a time, rendering one frame for every simulation step. This slows the simulation down, as the rendering of large systems takes a considerable amount of time. When the two tasks are assigned to different GPUs, then the simulation can run independently from the rendering, and the rendering device simply skips a step if it can not keep up with the simulation. However, using two devices comes at the cost of having to copy the current spin states from the simulation device to host memory and then onto the rendering device. Unfortunately, it is not possible to utilise NVIDIA's scalable link interface (SLI) for such a task, which would make it possible to by-pass the host.

If speed is of paramount importance, then the `colours_kernel_3d` described in Algorithm 2 can be made part of the simulation kernels for the single-GPU implementation, thus avoiding some overhead like the re-computation of cell coordinates. This was not done for this paper to preserve code readability by keeping the simulation and visualisation code separated.

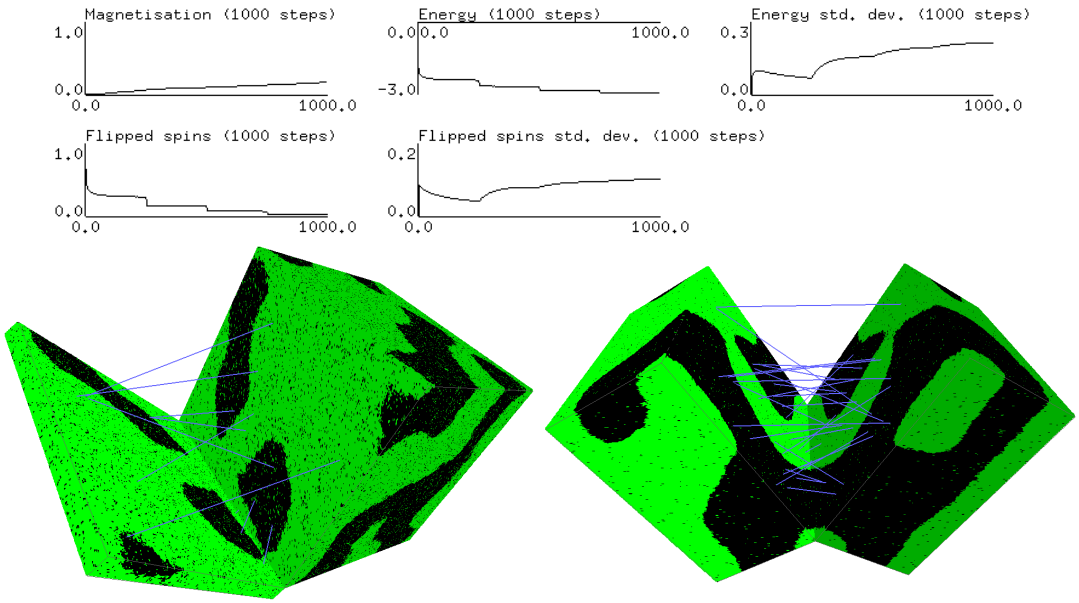

Figure 3: 3D Metropolis Ising model simulation with 256^3 cells, rewiring probability $p = 10^{-6}$ after 500 (left) and 1000 (right) simulation steps. The temperature is initialised to $T = 4.0$ and reduced by 0.5 every 250 simulation steps (i.e. $T = 3.5$ at 250 steps, $T = 3.0$ at 500 steps and $T = 2.5$ at 750 steps). The plots show the system state changes for the first 1000 simulation steps.

5. Conclusions and Future Work

We have shown before [27] that GPUs are excellent cost effective accelerator platforms for Ising simulation work due to their high number of homogenous cores and the consequent data-parallel programming model which is well suited to this task. The CUDA-OpenGL interoperability presented in this paper makes it possible to create high-performance visualisations for these simulations as the bulk of the data can stay on the device and never has to be copied between host and device memory. This has been demonstrated by means of the cell's spin values, which are determined by the simulation running on the graphics device, further used by the GPU to write to a VBO—which determines the vertex colours—and eventually used to render the cells.

We have also demonstrated how the real-time visualisation of simulations and system properties can improve the understanding of the behaviour of evolving complex systems. It can give new insights into these models to look at them at higher resolutions, which is possible due to the processing power of todays GPUs, since the multi-length scale properties of critical phenomena do show up better when there are several logarithmic scales to examine. It allows scientists to watch as phase transitions occur and enables them to modify system parameters while the simulation is running to get an immediate feedback of the consequences of these changes. Systems with small-world properties or other complex structures can be hard to understand scientifically and therefore benefit particularly from visualisation.

The ideas presented in this paper can be applied to other models and scientific simulations, such as the simulation of fluid dynamics and other aspects of materials science. Being able to peel away layers to view the interior of a 3-dimensional system can be very useful to get a feeling for what is happening.

We intend to further optimise the CUDA implementation of Wolff's cluster update algorithm in our future work. We are also planning to add additional visualisation and system analysis options that can help to better understand what is happening in the simulated system.

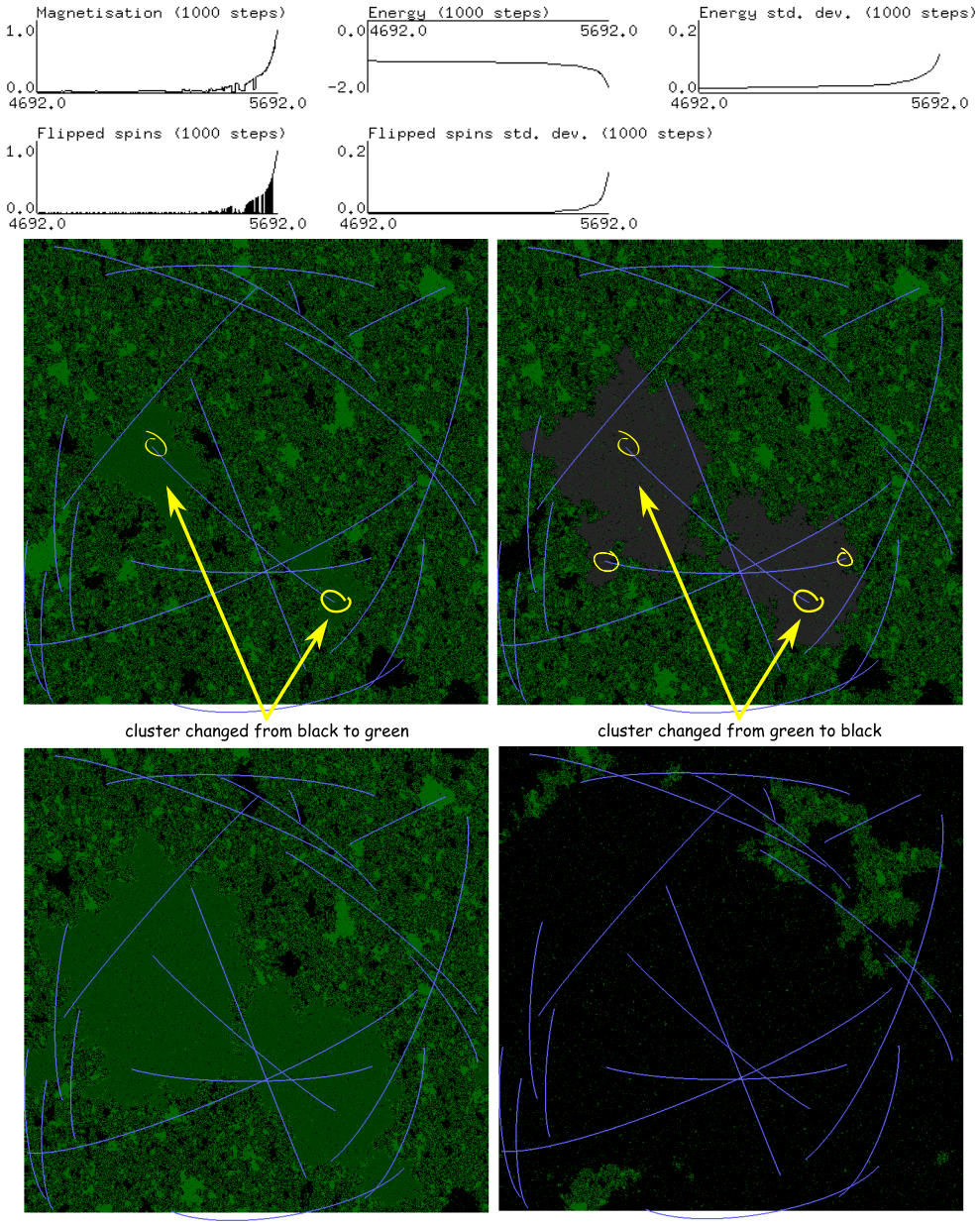

Figure 4: Ising simulation using Wolff's cluster update with 1024^2 cells, rewiring probability $p = 10^{-5}$ and temperature $T = 1.8$. The images show the simulation after 5461 (top left), 5582 (top right), 5632 (bottom left) and 5692 (bottom right) steps. The plots show the system changes of the last 1000 simulation steps. The circles in the first two images highlight the endpoints of the rewired edges which join the two otherwise disconnected clusters. The darker green (top left) and grey (top right) highlight the cells that had their spins flipped to green and black respectively during the previous simulation step. The plots emphasise the much more abrupt state changes of the Wolff cluster update as compared to the Metropolis algorithm.

References

[1] Y. Pan, R. Whitaker, A. Cheryauka, D. Ferguson, Feasibility of GPU-assisted iterative image reconstruction for mobile C-arm CT, in: Medical Imaging 2009: Physics of Medical Imaging, Vol. 7258, SPIE, 2009, pp. 72585J–1.

[2] R. A. Drebin, L. Carpenter, P. Hanrahan, Volume rendering, in: ACM SIGGRAPH Computer Graphics, Vol. 22, 1988, pp. 65–74, iSSN:0097-8930.

[3] J. D. Foley, A. van Dam, S. K. Feiner, J. F. Hughes, Computer graphics: principles and practice (2nd ed.), Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.

[4] T. Ritschel, Fast GPU-based Visibility Computation for Natural Illumination of Volume Data Sets, in: Eurographics (Short Papers), 2007, pp. 57–60.

[5] S. Bruckner, E. Groller, Enhancing Depth-Perception with Flexible Volumetric Halos, IEEE Trans. Visualization and Computer Graphics 13 (6) (2007) 1344–1351, ISSN:1077-2626.

[6] M. Schott, V. Pegoraro, C. Hansen, K. Boulanger, K. Bouatouch, A directional occlusion shading model for interactive direct volume rendering, Computer Graphics Forum 28 (3) (2009) 855–862.

[7] M. Meyer, R. M. Kirby, R. Whitaker, Topology, accuracy, and quality of isosurface meshes using dynamic particles, IEEE Trans. Visualization and Computer Graphics 13 (6) (2007) 1704–1711.

[8] E. Catmull, A. R. Smith, 3-d transformations of images in scanline order, SIGGRAPH Comput. Graph. 14 (3) (1980) 279–285. doi:http://doi.acm.org/10.1145/965105.807505.

[9] A. R. Sanderson, C. R. Johnson, R. M. Kirby, Display of vector fields using a reaction-diffusion model, in: Proc. Conf on Visualization'04, no. ISBN:0-7803-8788-0, ACM SIGGRAPH, 2004, pp. 115–122.

[10] D. Playne, K. Hawick, Visualising vector field model simulations, in: Proc. 2009 International Conference on Modeling, Simulation and Visualization Methods (MSV'09) Las Vegas, USA., no. CSTN-074, 2009.

[11] L. Smarr, C. E. Catlett, Metacomputing, Communications of the ACM 35 (6) (1992) 44–52.

[12] W. Bethel, B. Tierney, J. Lee, D. Gunter, S. Lau, Using high-speed wans and network data caches to enable remote and distributed visualization, in: Proc. of the 2000 ACM/IEEE conference on Supercomputing, 2000, pp. 1–23.

[13] R. Ng, B. Mark, D. Ebert, Real-time programmable volume rendering (2009).

[14] P. A. Fletcher, P. K. Robertson, Interactive shading for surface and volume visualization on graphics workstations, in: VIS '93: Proceedings of the 4th conference on Visualization '93, IEEE Computer Society, Washington, DC, USA, 1993, pp. 291–298.

[15] G. Vézina, P. A. Fletcher, P. K. Robertson, Volume rendering on the MasPar MP-1, in: VVS '92: Proceedings of the 1992 workshop on Volume visualization, ACM, New York, NY, USA, 1992, pp. 3–8. doi:http://doi.acm.org/10.1145/147130.147138.

[16] N. Goodnight, C. Woolley, G. Lewin, D. Luebke, G. Humphreys, A multigrid solver for boundary value problems using programmable graphics hardware, in: Proc. ACM SIGGRAPH/EUROGRAPHICS Conf. on Graphics Hardware, no. ISBN ISSN:1727-3471, San Diego, California, USA., 2003, pp. 102–111.

[17] I. Poupyrev, S. Weghorst, M. Billinghurst, T. Ichikawa, A framework and testbed for studying manipulation techniques for immersive VR, in: VRST '97: Proceedings of the ACM symposium on Virtual reality software and technology, ACM, New York, NY, USA, 1997, pp. 21–28. doi:http://doi.acm.org/10.1145/261135.261141.

[18] A. R. Sanderson, M. D. Meyer, R. M. Kirby, C. R. Johnson, A Framework for Exploring Numerical Solutions of Advection-Reaction-Diffusion Equations Using a GPU-Based Approach, Computing and Visualization in Science 12 (2009) 155–170.

[19] M. Rumpf, R. Strzodka, Nonlinear diffusion in graphics hardware, in: Proceedings of EG/IEEE TCVG Symposium on Visualization (VisSym '01), 2001, pp. 75–84.

[20] J. Bolz, I. Farmer, E. Grinspun, P. Schröoder, Sparse matrix solvers on the GPU: conjugate gradients and multigrid, ACM Trans. Graph. 22 (3) (2003) 917–924. doi:http://doi.acm.org/10.1145/882262.882364.

[21] J. Beyer, GPU-based Multi-Volume Rendering of Complex Data in Neuroscience and Neurosurgery, Ph.D. thesis, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria (Oct. 2009).
URL `http://www.cg.tuwien.ac.at/research/publications/2009/beyer-2009-gpu/`

[22] NVIDIA® Corporation, CUDA™: Compute Unified Device Architecture, `http://www.nvidia.com/` (2009).

[23] E. Ising, Beitrag zur Theorie des Ferromagnetismus, Zeitschrift fuer Physik 31 (1925) 253258.

[24] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. . Teller, E. Teller, Equation of state calculations by fast computing machines, J. Chem. Phys. 6 (21) (1953) 1087–1092.

[25] R. Glauber, Time dependent statistics of the ising model, J. Math. Phys II 228 (4) (1963) 294–307.

[26] U. Wolff, Collective Monte Carlo Updating for Spin Systems, Phys. Lett. 228 (1989) 379.

[27] K. Hawick, A. Leist, D. Playne, Regular Lattice and Small-World Spin Model Simulations using CUDA and GPUs, Tech. Rep. CSTN-093, Computer Science, Massey University (2009).