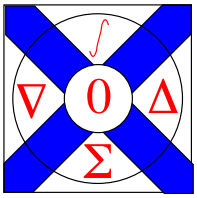

Computational Science Technical Note **CSTN-142**

Software Engineering a Family of Complex Systems Simulation Model Apps on Android Tablets

V. Du Preez and B. Pearce and K. A. Hawick and T. H. McMullen

2012

Tablet computers are emerging as powerful platforms for educational and demonstration software in areas like computational science and simulation which previously had needed higher performance processing. Developing software as an App or even porting it from other interactive platforms still requires non-trivial software engineering effort however. We descibe how a family of complex systems simulation models were developed as a domain related family of Apps and discuss the softwar e engineering issues we encountered in generalising the data structures, and simulation code patterns to run as Android Apps. We also discuss performance achieved on various models with a range of modern tablet computers and other devices with similar processors. We speculate on how domain-specific software engineering methods could automate such future simulation model App development.

Keywords: software architecture; tablet computing; Apps; Android; ARM processor

BiBTeX reference:

```
@INPROCEEDINGS{CSTN-142,
        author = {V. Du Preez and B. Pearce and K. A. Hawick and T. H. McMullen},
        title = {Software Engineering a Family of Complex Systems Simulation Model
                Apps on Android Tablets},
        booktitle = {Proc. Int. Conf. on Software Engineering Research and Practice (SERP'12)},
        year = {2012},
        pages = {215-221},
        address = {Las Vegas, USA},
        month = {16-19 July},
        organization = {SERP12-authors.pdf},
        publisher = {CSREA},
        institution = {Computer Science, Massey University},
        keywords = {software architecture; tablet computing; Apps; Android; ARM processor},
        owner = {kahawick},
        timestamp = {2012.05.03}
}
```

Technical Report CSTN-142

Software Engineering a Family of Complex Systems Simulation Model Apps on Android Tablets

V. Du Preez, B. Pearce, K.A. Hawick and T.H. McMullen
Computer Science, Institute for Information and Mathematical Sciences,
Massey University, North Shore 102-904, Auckland, New Zealand
{ dupreezvictor, brad.pearce.nz }@gmail.com, { k.a.hawick, t.h.mcmullen }@massey.ac.nz
Tel: +64 9 414 0800 Fax: +64 9 441 8181

March 2012

ABSTRACT
Tablet computers are emerging as powerful platforms for educational and demonstration software in areas like computational science and simulation which previously had needed higher performance processing. Developing software as an App or even porting it from other interactive platforms still requires non-trivial software engineering effort however. We describe how a family of complex systems simulation models were developed as a domain related family of Apps and discuss the software engineering issues we encountered in generalising the data structures, and simulation code patterns to run as Android Apps. We also discuss performance achieved on various models with a range of modern tablet computers and other devices with similar processors. We speculate on how domain-specific software engineering methods could automate such future simulation model App development.

KEY WORDS
software architecture; tablet; Apps; Android; ARM.

1 Introduction

Engineering software for tablet computers is an exciting area of development with emerging framework software proving capabilities to exploit the mobile nature and touch screen interaction capabilities of such devices. Despite a somewhat tumultuous history [1, 2], these devices have already found uses in mobile gaming [3–5] and other computationally intensive applications beyond simple data entry [6, 7].

These platforms are here to stay and have a strong potential [8]. In this paper we describe our interests

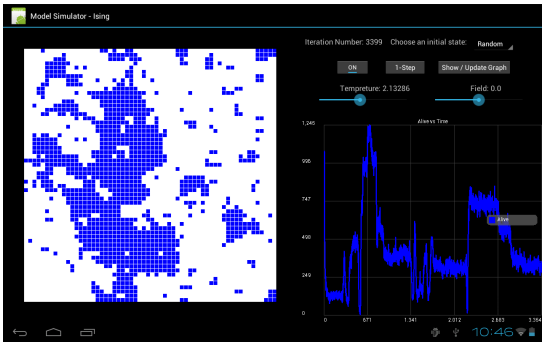

Figure 1: Screen-dump showing our App running the Ising Model

in developing software for computational interactive simulations [9] and our efforts in engineering simulation Apps for tablet computers using the Android operating system [10–12] and associated code frameworks.

Tablets are powerful platforms for simulation demonstration models, due to their highly interactive graphical display capabilities. However coding for this interactivity can be demanding due to the still maturing touch and multi touch interfaces. Tablet architecture is becoming more and more advanced with the development of duel core and recently quad core systems. This allows for models be become more interactive and become more graphically intense.

Mobile operating systems, such as Apple's IOS and Googles Android have provided developers a solid platform to implement classical models on to test both the devices and the models as interaction with the model is made easy. Stumbling blocks are present however when moving a model from a PC to a mobile operating system. Considerations such as power man-

agement, memory management and conservation, and limited computational power all need to be considered when implementing a chosen model.

The Android platform allows a somewhat self managed system for its Apps due to the use of the Dalvik virtual machine [13]. This is a type of Java virtual machine which is much like Sun's JVM but has optimizations in the areas of power management, better memory sharing and other aspects which enable strong performance on mobile devices such as phones and tablets.

Todays tablets and mobile devices have Advanced Risk Machine (ARM) processors [14]. This is as these ARM processors are low powered and cheap to make which allows fast and efficient mobile computing in comparison to more powerful and power hungry PC CPUs. Even though ARM processors are now as powerful as PC processors they are still capable of running the latest mobile operating systems of IOS and Android. Many devices have multiple ARM processors which perform different aspects for the tablet or mobile device.

Multiple ARM processors and other hardware can be formed into an integrated circuit know as a System on Chip(SOC). A SOC can contain all the hardware that is required to operate a tablet or mobile device, hardware such as multiple ARM processors for applications, GPS, blue-tooth, WiFi and RF communications.

The SOC also contains many busses that link all the processors and many other bits of hardware for peripheral devices for the tablets. There is a limited number of companies that manufacture SOCs which are known for slight optimizations in certain areas of the various chips. SOC however are not limited to tablets and mobile devices, there are various organisations that have used SOCs in various other development boards such as Panda [15] and Raspberry Pi [16].

As both ARM processors and SOCs become more complex and powerful this allows for tablets and other mobile devices to take advantage of portable and interactive computing power which allows for interesting research in App development. Tablets have been optimized with version of an ARM architecture processor for longer battery life vs battery weight.

ARM gained popularity with more power and cost efficient way to build tablet or mobile computers, running both and Android and IOS. The touch display capability supports direct model manipulation of complex systems such as the: Ising model of a magnet [17–20]; Game of Life [21, 22]; Game of Death [23, 24]; Kawasaki exchange model [25, 26]; and Sznajd opinion model [27, 28] which we discuss in Section 2. Figure 1 shows a screen-shot from our App running the Ising model simulation.

In Section 3 we describe various aspects of Android's App architecture that we exploited to engineer our Apps the platform works. In Section 4 we explain our results from the various models and we discuss this further in our discussion in Section 5. We offer some conclusions and directions for further work in Section 6.

2 Model Family

The driving motivation for our building a family of simulation model Apps was our desire to demonstrate these to students learning about the models [29], coupled with the realisation that many of these models have a great deal in common. In software engineering terminology they constitute a family of domain models that are closely related. We found that it was feasible and desirable to construct our App software to exploit this observation.

The models themselves have been described extensively elsewhere - both by ourselves and by other authors. However we give a brief summary of the key features here for completeness and to explain the decisions and results discussed in this present article.

The Game of Life and Game of Death are cellular automaton models. Each cell is initialised to a state or live, dead (or zombie) and a deterministic rule is applied to change the state according to its own state and that of its immediate neighbours. These models are surprisingly complex systems whereby rich and unexpectedly structured spatial patterns of cells emerge from the very simple microscopic rules applied deterministically to the individual cells.

The Ising model is almost as simple a model. In it, a heat-bath algorithm is used to emulate thermal effects on atoms in a magnetic material arranged in a crystalline lattice. The Ising system consists of a micro crystalline array of single bit magnetic moments or "spins" which interacts with its nearest neighbours. At each time step of the simulation each spin is considered in turn and the energy and thermal probability of it "flipping" – reversing its direction are considered. The probability of flipping is different, depending upon the applied temperature.

We find that spins align with their neighbours when the system is cold, but thermally randomize when it is hot. The interesting feature about the Ising system

in 2 (or 3) dimensions is that there is a definite Curie temperature that can be measured. In real magnets the Curie temperature is the temperature above which the material stops being a magnet, or an alternative viewpoint is that materials like iron spontaneously become magnetic below their Curie temperature. This is known as a phase transition and is very difficult to explain simply without a model to demonstrate. The value of an interactive simulation is that the temperature parameter can be directly varied by the user and the effects seen in real time. Thus, it is possible for a user of our App to cool or quench a simulated Ising magnet down and see it undergo its phase transition and all the spins start to spontaneously align with one another to produce complex patterns.

The Kawasaki exchange model is constructed in a similar manner to the Ising system. In this case however we preserve a fixed ratio of the two microscopic species since instead of flipping or changing species, in the Kawasaki system we only allow them to swap positions with one of their (randomly chosen) neighbours. In this respect the Kawasaki system models diffusion and phase separation or "unmixing" of the two species. the rate and manner of unmixing is like the separation of two atomic species in a binary alloy. This sort of dynamical behaviour is of great importance in real materials. With out some separated granules an alloy typically lacks strength and other physical properties but if too much separation occurs it can break apart and cause catastrophic failure in for example fuel rods in a reactor.

Models of opinion formation and propagation are now recognised as important ways to understand crowd behaviours, social and political phenomena. The Sznajd opinion model is also formulated in a manner similar to the Ising or Kawasaki models, but the update rules are simpler. Each species represents an opinion, and at each step if two cells of the same opinion are found next to one another we allow the possibility of them "persuading" all their neighbouring sites to the same opinion. In this sense opinions get pushed out to the neighbours via the Sznajd update rule.

We encounter similar complex spatial patterns formed by all these models. They are all modelled as an array of sites on for example a square lattice. Each cell can take on a discrete and small number of different states – live/dead/zombie or spin-up/spin-down or alloy species x/y, or political opinions A/B/C and so forth. An array of bytes is suited to this model. The update rules are only slightly different for each model so some commonality in code structure is possible.

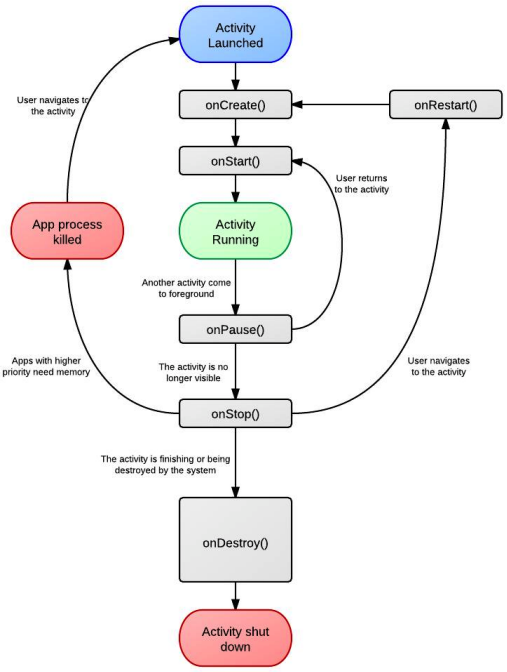

Figure 2: Android Activity Life-cycle

For convenience in applied the update rules symmetrically we impose periodic boundary conditions - so that a cell at the edge of the array still sees the same number of neighbouring cells as one in the middle. This does not affect the physics or complexity of the model and also supports visualisation actions such as panning and zooming around in a wrap-around display of the simulated system.

The key criteria for engineering a family of Apps for these models are the performance capabilities and managing the code complexity to implement suitable model system sizes on current generation tablet computers using the code development frameworks available for them.

3 Android Java Apps Architecture

The Android operating system has emerged as a powerful and popular platform for developing Apps to run on mobile platforms such as tablet computers and smart phones. Android is a Unix/Linux-like operating system with many well established, tested and widely used and understood internal software models. Its relative openness makes it an attractive development platform for mobile applications.

Figure 2 shows the Android activity life-cycle. It

shows the different calls an Android activity makes on its creation, halt and restart. Our application used many of these activity call-backs to set up and run our simulations. We mainly used the onCreate() function to initialize data and classes and the onPause() function to halt the simulations when the activity was not longer in view by the user using the home button or the back button.We also used the onRestart() call-back to restart our App if the user navigated back to the application. This life-cycle call-back functions of the activity are extremely useful in application as they are mainly event driven. However as can be seen the system does the handling of properly destroying the activity which means from a programming point of view you need to be careful about how you exit and clean up in your application.

```
runSimulation function{
new Thread {
  While (active){
    if (redraw){
        calluiupdate()
    }else if (Running){
        choose rule set
        calluiupdate()
    }else if (one step){
        choose rule set
        calluiupdate()
    }
  }
}
```

Figure 3: Pseudo code for main loop

The code listed in Figure 3 is an example of a running model in the main loop. This is pseudo-code for creating new thread controlling the main loop and handling the invalidation for the GUI. The code stipulates that when the activity is active update the model in a continuous loop otherwise a one-step mode can be performed which updates each step individually with button control.

Figure 4 and Figure 5 show pseudo code for our rule implementation of the Game of life and the Game of Death. For the game of life we loop through all the cells. For each one we get the sum of the number of live cells surrounding the current cell and then apply the rules to the cell bases on this count. The rules are implemented in the if statements. After every cell has been updated this is considered one step or iteration in the game of life.

For the Game of Death in Figure 5 the implementation of the rules was slightly more complex due to the addition state of zombie but it is very similar to the

```
golconwayrules(){
get number of cells in grid;
  for(int i=0;i<totalcells;i++){
        count num of live neighbours;
        if ((state == 0) && (count == 3)){
            set alive;
        }else if ((state == 1) &&
           ((count == 2) || (count == 3)))
            set state to alive;
        }else{
            state dies;
      }
  }
}
```

Figure 4: Pseudo code for Game of life computation

```
public static void godrules(){
  get number of cells in grid;
  for(int i=0;i<totalcells;i++){
      count num of live neighbours;
     if state alive and
         count is 3 or count is 2{
       then set alive state;
     }else if  state is alive{
       set to dead state;
     }
     if state is dead and count is 3{
       set to alive state;
     }else if state is 0){
        set state to zombie;
     }
     if state is zombie and count is 2{
       set state to alive;
     }else if state is zombie{
       set state to zombie;
    }
  }
}
```

Figure 5: Pseudo code for Game of life computation

game of life. Again we loop through all the cells, for each we need to know its current state and how many are alive around it. We can then apply the rules in the if statements to the cell to produce the next state. When all cells have been updates we again consider that one iteration.

The App architecture can derived from Figure 2 showing the App runtime environment. Android supports what is know as a multiple activities model and the screen area is managed using various ”activities.” We found it useful creating multiple activities to navigate through the application selecting and initializing each model. Once activity is created it can send Intents to the Android system which starts other activities. This

Device	FPS	Time(s)
Motorola Xoom	26	38.22s
Samsung Galaxy Tab	34.8	28.95
Asus Transformer	28.2	35.58s
Panda Developer board	17	59.69s
Android Emulator	4.6	213.57s
Animaux (PC Build)	24.7	40.5s

Table 1: Averages for 1000 iterations for Game of Life

allows us to combine loosely couple components to perform certain tasks.

Graphical Apps creates a single thread of execution by the Android OS which is called the main thread. UI, invalidation and canvas drawing are performed on this thread and we implemented this with the code on Figure 3, creating a new thread for our drawing to the canvas. Developers can not access the main thread directly, however AsyncTask can be used for thread management. Manipulating thread and handlers are avoided when the UI thread controls background operations and result.

These models can be defined in a arbitrary dimension most commonly described as "hyper bricks" of mesh points. This allows us to have a block of memory to index any point in the brick using a single integer which is known as the "k-index" [30]. In our implementation both a k-index and x coordinate system was used due to X,Y location coordinates being precomputed and stored in an array rather than converting to X and Y from a k-index. This was to reduce the amount of modulus operations in the code which has been known reduce performance in computational models. The negative aspect of precomputing the X and Y coordinates is that it requires additional storage for X,Y coordinates on top of storage of the state of each cell.

4 Performance Results

We have measured the simulation and graphical update performance of our model Apps on various specific tablet and other compute platforms.

Table 1 , Table 2 ,Table 3 and Table 4 all show interesting results when compared to each other. The tested tablets, Motorola Xoom, Samsung Galaxy Tab and the Asus Transformer all showed similar performance. This was expected as they all contain the same hardware (See Table 5). This was apparent when comparing the Xoom and the Transformer tablet devices. The Galaxy Tablet however performed much better than expected in comparison to the other two

Device	FPS	Time(s)
Motorola Xoom	15.2	69.16s
Samsung Galaxy Tab	18.8	55.58s
Asus Transformer	17	58.63s
Panda Developer board	11	95.16s
Android Emulator	4.6	213.57s
Animaux (PC Build)	24.7	40.5s

Table 2: Averages for 1000 iterations for Game of Death

Device	FPS	Time(s)
Motorola Xoom	24.2	41.30s
Samsung Galaxy Tab	31	32.41s
Asus Transformer	26.2	38.40s
Panda Developer board	18.2	58.04s
Android Emulator	4.4	274.15s
Animaux (PC Build)	24.7	40.5s

Table 3: Averages for 1000 iterations for Ising

tablets. We can only speculate as to why this occurred. The Panda Developer board performed slower than the tablet devices which was surprising as it is very similar in hardware.

The Android emulator that is provided with the Android SDK performed the worst of all tested devices. We suspect this to be as all the hardware of the emulator is emulated in software unlike the other devices that run on physical hardware. The Animaux program for PC performed the same across the all the models. This was expected as the program is programmed in a totally different way with different classes and underlying classes and data structures. Therefore it make it hard for us to draw any relevant conclusions as to how powerful PC simulations are compared to those on tablet based devices.

Table 5 shows each device specification that these complex models was computed on. Android tablets and the Panda Developer board running the Dual Core Cortex A-9 chip from the ARM family, this was somewhat interesting since similar results are

Device	FPS	Time(s)
Motorola Xoom	87.8	11.50s
Samsung Galaxy Tab	128.4	7.77s
Asus Transformer	78.8	12.73s
Panda Developer board	88.6	11.40s
Android Emulator	6.8	153.87s
Animaux (PC Build)	24.7	40.5s

Table 4: Averages for 1000 iterations for Sznjad

Device	CPU	Memory	GPU	L2/L3 Cache
Motorola Xoom	NVIDIA Tegra 2 T20	1 GB - DDR2	ULP GeForce	1MB - L2
Samsung Galaxy Tab	NVIDIA Tegra 2 T20	1 GB - DDR2	ULP GeForce	1MB - L2
Asus Transformer	NVIDIA Tegra 2 T20	1 GB - DDR2	ULP GeForce	1MB - L2
Panda Developer board	Dual-core Cortex-A9	1 GB - DDR2	SGX540 graphics core	1MB - L2
Android Emulator	NA	256Mb - VM	NA	NA
Animaux (PC Build)	3.2 GHz Intel Core i3	4 GB DDR3	ATI Radeon HD 5670 512 MB	4MB - L3

Table 5: Specifications for the computer platforms used.

expected between Android devices. These Android tablets are identical and the only difference to similar devices is the Graphics Processing Units (GPU) in the Panda Development board. Android tablets use a 8 core 333 MHz GPU whereas the Panda boards use 4 core 384 MHz both supporting OpenGL ES 2.0.

Results shown the Samsung device having much faster refresh rates and lower time steps compared to other tablet devices. Since all the models had the Android Ice Cream Sandwich 4.0.3 running as their main operating system, we concluded differences in driver, memory and resource management between the different vendors. Android emulator showed slow results due to the ARM architecture that Android is based on. Personal computers are based on the x86 architecture and thus the emulator need to be executed on a virtual machine which was irrelevant to the test we wanted to contrive on Android OS.

5 Discussion

Our test environments which were all dual core architecture (Tegra 2). This allows creation of a genuinely concurrent new thread so that calculations of the rules and Android's output on the display can be separated. This provided significant performance increases. (see performance comparison graph/grid). This is as Android/Java's system, garbage collection and user display processes are on one core and the other core is left to do all the calculations on the rule-set. When the newer quad core architecture is available (Tegra 3), the App may then be able to run the various models a lot faster as it splits the work load across the four cores the cpu provides.

Models showed that similarities such computation time and refresh rate could be closely correlated, however the Game of Death showed these variables to be slower in all aspects. Having three states was display intensive on the model with garbage collection(GC) initially running at 40%, this was optimized using the Android Debug Bridge(adb). This was a crucial tool developing the App showing thread usage and garbage collection.

The various models across different devices and platforms gave us many interesting and unexpected results. Firstly the various models, Game of life, Game of Death, Ising, and Sznajd all produced interesting results across the three tablet devices. As can be seen from table one through four all the tablets showed slight performance differences. This was interesting as all the tablets are based on the same duel core, Tegra 2 architecture. It was surprising to see the rather large variation in the amount of time it took to do 1000 steps in the various models across the Xoom, Transformer and Galaxy Tablet. The only thing we can put these variations are down to the specific hardware differences of the three tablet devices and the device drivers associated. This could also be due to different ARM processor versions, cache, clock speeds, stand alone GPU processors in devices.

The FPS of the various devices varies in direct relation to the time taken to perform the 1000 iterations of the model. As FPS rises the time taken decreases. This was due to the multi-threaded nature of the application and how Android platform separates its processes across its various available cores. The Panda development board fared somewhat worse than expected in testing. This may be due to its unoptimized nature to the Android platform as the panda board is also designed to run various other operating systems. The Android emulator that comes with the Android SDK performed slowest of all across all the models. This we suspect is due to the emulator running via software rather than hardware like on the tablets. This may be causing a bottleneck in the emulator in graphic output.

The PC based Animaux model performed uniformly across all models in terms of iteration times and FPS. This performed well and it was obvious that the PC implementation had a lot more possible power that could be squeezed out of it if optimized to the models such as the Android App had. Even though the Animaux model was also written in Java, underlying

classes and grid structure were used. This means differences in neighbour discovery, state recovery. Differences in 1000 iteration times in the four models was due to what neighbours where needed by the various models and the requirements or computing them. As these models are loop intensive resource reuse was something we needed to be wary of as garbage collection by the Dalvik virtual machine can be time intensive. The Sznajd model requires the least amount of neighbours to be retrieved hence it performs the fastest.

6 Conclusions

We have described how we architected and developed a family of Apps for tablet computers using Android and associated Java frameworks. We have shown that modern tablet devices produce quite credible performance even when compare d to a desk top implementation - for interactive simulation demonstrations. We have discussed software and implementation commonalities across our initial family of application simulation models - including simple automata such as the Game of Life and Game of Death through more sophisticated automata-like models such as the Sznajd opinion models to models like the Ising and Kawasaki models which require random number generation and mathematical function evaluations.

The Android code development framework is quite well suited to such a family of Apps and we have been able to share a lot of code and data structures across this family of models. The graphical interface code and associated idiomatic structures are rather different for a touch sensitive screen found on the tablet to the more common mouse environment and widget collection used on desktops but it is feasible to find ways for a user to exploit either.

The Android activities model and the associated lifecycle is a good framework for the interactive nature of Android applications. It lends itself well to the event driven flow and style of direct model manipulation in computational models. The activity design with callback functions enables users to prioritize and handle views and the underlying computations in a clean and theoretical way. It also allows easy instantiation and destruction of activities and associated objects that the activity needs before and after the running of the activity. This aspect of the Android system allowed us to handle and optimize our activities and the flow of our activities to enable our App to be as intuitive as possible.

An interesting area for future development will be to find ways to combine these model generation ideas so that code can be generated for either a tablet or a desktop platform. We have made some inroads into exploring the commonalities across our target simulation models and algorithms, and have identified ways for them to share data structures. There is still considerable work to be done however to identify a common software model architecture for the user interaction and parameters control framework that would be needed to fully automatically generate Desktop and App codes.

References

[1] Yarow, J.: Tablet computing: A history of failure. Business Insider **Online** (2010) 1–3

[2] Norman, D.A.: Inside risks: Yet another technology cusp: Confusion, vendor wars, and opportunities. Communications of the ACM **55** (2012) 30–32

[3] Cheng, K.W.: Casual gaming. VU Amsterdam (2011)

[4] Kemp, R., Palmer, N., Kielmann, T., Bal, H.: Opportunistic communication for multiplayer mobile gaming: Lessons learned from photoshoot. In: Proc. Second Int. Workshop on Mobile Opportunistic Networking (MobiOpp'10), Pisa, Italy (2010) 182–184

[5] Feijoo, C., Ramos, S., Gomez-Barroso, J.L.: An analysis of mobile gaming development - the role of the software platforms. In: Proc. Business Models for Mobile Platforms(BMMP10), Berlin, Germany (2010)

[6] Buchanan, N.: An examination of electronic tablet based menus for the restaurant industry. Master's thesis, University of Delaware (2011)

[7] Castellucci, S.J., MacKenzie, I.S.: Gathering text entry metrics on android devices. In: Proc. Computer Human Interactions (CHI2011), Vancouver, BC, Canada (2011) 1507–1512

[8] Coulter, R.: Tablet computing is here to stay, and will force changes in laptops and phones. Mansueto Ventures (2011)

[9] Huynh, D.B.P., Knezevic, D.J., Peterson, J.W., Patera, A.T.: High-fidelity real-time simulation on deployed platforms. Computers & Fluids **43** (2011) 74–81

[10] Conti, J.P.: The androids are coming. Engineering and Technology Magazine **May - June** (2008) 72–75

[11] Johnson, M.J., K. A, H.: Porting the google android mobile operating system to legacy hardware. In: Proc. IASTED Int. Conf. on Portable Lifestyle Devices (PLD 2010), Marina Del Rey, USA (2010) 620–625

[12] Kim, S.: Logical user interface modeling for multimedia embedded systems. In: Proc. Int. Conf on Multimedia, Computer Graphics and Broadcasting (MulGrab 2011), Jeju Island, Korea (2011)

[13] Ehringer, D.: The dalvik virtual machine architecture. Technical report, Google (2010)

[14] Sloss, A.N.: ARM System Developer's Guide: Designing and Optimizing System Software. Elsevier (2010) ISBN 978-0080-490-496.

[15] Panda: Panda board development resources (2012)
[16] Pi, R.: Raspberry pi - an arm gnu/linux box (2012)
[17] Ising, E.: Beitrag zur Theorie des Ferromagnetismus. Zeitschrift fuer Physik **31** (1925) 253?258
[18] Hawick, K., Leist, A., Playne, D.: Cluster and fast update lattice simulations using graphical processing units. Technical Report CSTN-104, Computer Science, Massey University (2009)
[19] Leist, A., Playne, D., Hawick, K.: Interactive visualisation of spins and clusters in regular and small-world Ising models with CUDA on GPUs. Journal of Computational Science **1** (2010) 33–40
[20] Hawick, K.A.: Domain Growth in Alloys. PhD thesis, Edinburgh University (1991)
[21] Gardner, M.: Mathematical Games: The fantastic combinations of John Conway's new solitaire game "Life". Scientific American **223** (1970) 120–123
[22] Hawick, K.: Cycles, diversity and competition in rock-paper-scissors-lizard-spock spatial game simulations. In: Proc. International Conference on Artificial Intelligence (ICAI'11), Las Vegas, USA (2011)
[23] Resnick, M., Silverman, B.: Exploring emergence: The brain rules. `http://llk.media.mit.edu/projects/emergence/mutants.html` (1996) MIT Media, Laboratory, Lifelong Kindergarten Group.
[24] Hawick, K., Scogings, C.: Cycles, transients, and complexity in the game of death spatial automaton. In: Proc. International Conference on Scientific Computing (CSC'11). Number CSC4040, Las Vegas, USA (2011)
[25] Kawasaki, K.: Diffusion constants near the critical point for time dependent Ising model I. Phys. Rev. **145** (1966) 224–230
[26] Hawick, K.: Visualising multi-phase lattice gas fluid layering simulations. In: Proc. International Conference on Modeling, Simulation and Visualization Methods (MSV'11), Las Vegas, USA (2011)
[27] Sznajd-Weron, K., Sznajd-Weron, J.: Opinion evolution in closed community. Int. J. Modern Physics C **11** (2000) 1157–1165
[28] Hawick, K.: Multi-party and spatial influence effects on opinion formation models. In: Proc. IASTED International Conference on Modelling and Simulation (MS 2010). Number CSTN-032, Calgary, Canada (2010) Paper 696-035.
[29] Fenwick, J.B., Kurtz, B.L., Hollingworth, J.: Teaching mobile computing and developing software to support computer science education. In: Proc. 42nd ACM Tech. Symp. on Computer Science Education SIGCSE'11, Dallas, Texas (2011) 589–594
[30] Hawick, K.A., Playne, D.P.: Hypercubic Storage Layout and Transforms in Arbitrary Dimensions using GPUs and CUDA. Concurrency and Computation: Practice and Experience **23** (2011) 1027–1050