2010

Using open-source platforms for digital media production

Brogan Bunt
University of Wollongong, brogan@uow.edu.au

Publication Details

Bunt, B. "Using open-source platforms for digital media production." Mobilizethis 2010. Charles Darwin University: Mobilizethis, 2010.

A Technical Framework for Introducing Computational Media Art Practice

A/Prof Brogan Bunt

Head of Postgraduate Studies

Faculty of Creative Arts

University of Wollongong

Introduction

Media art, like contemporary art generally, has a strong critical aspect. Media art practice involves a crucial dimension of interrogating the cultural, social-political and material-aesthetic conditions of media. From Dada photographic collage through to contemporary hardware hacking there is a clearly evident concern to unsettle the representational transparency and taken-for-granted character of media. How does this inform the teaching of media art? It has obvious thematic importance, indicating paths of conceptual access and orientation, but what of the dimension of practice? What of the technical frameworks that we employ and the skills that we teach? How do they obtain a critical inflexion? This paper focuses on an attempt to foster critically informed digital media practice within a first year undergraduate media art subject.

Computational Media is the second of two introductory media art subjects within the Bachelor of Digital Media and the Media Art minor at the University of Wollongong. The first subject, Introduction to Media Art, focuses on the conventional media of photography, audio and video. Computational Media explores the underlying technical layers of digital media. It aims to provide an accessible introduction to programming and electronics. Despite our best efforts, the subject tends to prove quite challenging for students - many of whom have a primarily audio-visual focus. They typically have the expectation that media art involves working with cameras, audio-recorders and an industry-standard set of proprietary software tools. Computational media aims to suggest other possibilities and to survey relevant traditions of critical media art. It draws inspiration from Lev Manovich's materialist conception of new media (Manovich, 2001), and Alan Kay's notion of computational media as a “metamedium” (Kay, 2003, p.394). At a broader level, the subject is informed by traditions of media theory that question the simple instrumental relation between human creative agency and *techne.* While the subject makes no effort to provide a detailed theoretical survey of the topic (it has, instead, a strongly practical studio focus), the writings of Heidegger, Benjamin, Adorno, McLuhan, Derrida, Kittler, Latour, Zielinski, Stiegler, etc. provide an important philosophical background. Overall, rather than envisaging the digital as an unproblematic space of bright and shiny new 'creative tools', the complex, awkward and labyrinthine relation between art and *techne* is emphasised.

The subject has three main aims:

1. To expose the computational dimensions of digital media and to find effective means for non-technical students to creatively engage with it. The focus is very much upon framing a set of compelling encounters with the field rather than attempting to develop dedicated programming or electronic skills.

2. To survey traditions of art that engage speculatively and critically (Fuller, 2003) with aspects of technological process (and computation specifically). This can involve considering artists as diverse as Duchamp, Tinguely, Xenakis and Matta-Clark, as well as any number of contemporary new media, software art and physical computing practitioners.

3. To introduce students to relevant contexts of contemporary practice. It is worth noting that these contexts often take shape around particular open-source hardware and software platforms.

There are a rich variety of technological frameworks available to pursue these aims. Having taught this kind of subject for the past fifteen years - having tried and occasionally failed to establish a good balance between depth and accessibility, technical encounter and creative-critical conceptualisation – I have gradually come to prefer specific methods and frameworks.

During the 1990s I tended to employ proprietary software environments such as Macromedia Director and Flash (now Adobe products). This worked well in terms of making code accessible for Media Art and Graphic Design students, but tended to constrain a genuinely creative relationship to the digital medium. Creative conceptions were fundamentally affected by the metaphors which shaped these packages. Furthermore, in their positioning as proprietary multimedia production tools, Director and Flash did little to connect students to relevant communities of practice.

I also spent some years teaching Java game programming to Informatics post-graduates. For me, this represented a sudden transition to a more intensive technical space. It also highlighted a difficult problem. Computer science students had the skills to engage with the material and immaterial complexity of digital media, but they lacked the conceptual background or motivation to explore the medium in a critically informed creative manner. In contrast, visual art, media and design students tended to withdraw from the technical space of code as though it were a creative desert. Neither group were quite appropriately positioned to pursue a genuinely experimental relation to the digital medium. This remains a problem. However, during the past decade an increasing number of Free/Libre/Open Source Software (FLOSS) projects have emerged that focus specifically on addressing this issue. These projects aim to provide free and accessible means for artists to engage with dimensions of computational process. Processing (www.processing.org) and Arduino (www.arduino.cc) provide the best known examples. Processing is a simplified Java-based integrated development environment (IDE) with an excellent help/reference system, a growing set of audio-visual focused libraries and a very active creative community (with a strong institutional base worldwide). Processing is refreshing because it strips away the traditional media focused metaphors of the proprietary multimedia software. The artist-programmer encounters instead a simple blank page. In this manner, writing, and the essential conceptual work that code authoring entails, is highlighted as the key point of creative focus. Arduino shares the same graphic user interface (GUI) as Processing but focuses on physical computing applications. It has a dual identity as both a C-based IDE and a micro-controller hardware platform.

I have taught with Processing and Arduino for several years and they provide a very good framework for teaching computational media. However they have an important weakness. This weakness is closely linked to their strength. In providing a supportive and accessible space for artists, they foster dependence and a resistance to exploring more mainstream languages and frameworks. While questioning the awkward cultural divide between technical and creative development systems, they also serve to indicate and partly reinforce key points of cultural distinction. One other significant problem is that they adopt the guise of discrete software programs. It is not as though creative computational media is enabled simply within specific applications. There is a more general and integral space that requires recognition. This takes emblematic form, for example, in the GNU/Linux Bash shell operating system interface which serves to mediate all kinds of user and code-based interaction with the underlying operating system. It is the traditional place, for instance, in which code (as a set of text files) is compiled and executed. If only as a means of appreciating relevant features of this interface - gaining an intimate, historically informed, grasp of the pre-GUI (or contra-GUI) environment - some experience working with the Bash shell/terminal/command line space can prove useful. This also, very importantly, involves engaging with the ethics and radical politics of FLOSS culture. In this manner, something as simple as the Bash shell interface can provide a concrete means of addressing issues of human/*techne* mediation, oppositional media practice, remix culture, etc. The technical framework of the GNU/Linux operating system can serve as an effective facilitator for critical-aesthetic engagement.

So in my planning for the 2010 version of Computational Media I was moving towards a system with the following features:

- It would be a holistic environment. More specifically, students would work with the GNU/Linux operating system. This would serve not only as an effective technical framework for creative computational practice but also as an alternative to mainstream proprietary operating systems and a focus for critique of the social relationships that proprietary systems embody.
- The emphasis would be on bash shell style interaction with the operating system, encouraging students to think beyond accepted GUI paradigms and empowering students to deal with fundamental programming processes and system level tasks.
- The selected programming language would be a modern, agile language. My preference was

www.scala-lang.org/) because of its clear and concise syntax and interoperability with Java.

- The physical computing platform would be Arduino due to its strong level of tutorial support and its wide community of practitioners.

Now it is all very well to describe a potentially effective technical infrastructure, but getting it installed on centrally managed machines in university teaching labs is another matter. There are all manner of good reasons why implementing a perfectly reasonable (and perfectly free) system may not feasible. For example, there may be a lack of suitably qualified support personnel, or the framework may not adequately integrate with existing university systems, or the perceived security risks may be too high. We quickly discovered that the challenge we faced in this project was not only at the level of technical design but also that of negotiating a place within the University of Wollongong IT system.

An Institutionally Friendly Technical Framework

During the latter part of 2009, an associate, Peter Goodall, and I obtained a $14,000 University of Wollongong Educational Strategies Development Fund grant to support the development and testing of a suitable technical framework. Peter is an experienced IT professional and a research fellow in the Digital Ecosystems Research Centre (UOW Informatics Faculty). His role has been to design the system infrastructure, liaise with Information Technology Services (ITS) staff and provide technical support for the pilot project. We also enlisted vital support from Sarah Lambert, who is a senior manager in ITS responsible for fostering and managing innovative IT projects. She has played a key role in getting the project formally approved.

We originally envisaged implementing the system in a restricted 'sandbox' environment. The aim was to install the operating system and all the associated software in two dedicated Digital Media/Media Arts labs. The system would run in pilot mode for the teaching of a single session of Computational Media. If the project proved successful then we could make a case for wider roll-out. This plan proved impractical. Since both labs were maintained by central ITS, we needed permission to install a new operating system (the GNU/Linux distribution, Ubuntu). Technical ITS support, however, was only available for the Windows and Macintosh operating systems. Even small-scale implementation would require employing additional technical staff, as well as considerable resources to research and test dimensions of overall system management and integration. We lacked the budget to support any of this.

The next option was to employ a live (USB bootable) GNU/Linux distribution. Pure Dyne (http://puredyne.org/) seemed an obvious choice since it is directed towards the needs of audio-visual artists and provides links to a wide community of media art practitioners. Live distributions, however, require that a lab machine can be re-booted from an external USB key. This represents a potential security threat and so is not permitted in our teaching labs.

Then Peter came up with the solution – virtualisation. Since we were not permitted to install an alternative operating system, why not run GNU/Linux on top of the existing operating system? All that was needed was the installation of virtual machine player on the host system and the GNU/Linux system could be launched from a USB drive. The important feature was that the virtual machine would never literally interact with the underlying operating system. Furthermore, when the system is closed, the virtual machine would disappears altogether, leaving no trace of its presence on the host system. The lack of dialogue with native registries and the like makes everything much more clean and simple. At the same time, the virtual system still retains the capacity to save and maintain its own internal state, so that students can save their work from one session to the next. Apart from its overall value in preserving aspects of system integrity and security, virtualisation offers two other key advantages: firstly, it enables students to keep a portable version of the overall technical framework for use at home, etc. without any risk of copyright breach and without any expense (apart from the purchase of a suitable USB key or hard disk); and secondly, it simplifies lab installation and maintenance for ITS staff, so that instead of a installing and supporting a plethora of obscure FLOSS software, they now only have to worry about managing the single virtual machine player.

The plan was to implement this framework for the Autumn 2010 running of Computational Media. Spring session was devoted to the technical development of the system and close liaison with ITS. As I mentioned earlier, the key breakthrough came when Sarah Lambert assisted us in positioning our ESDF project as a pilot ITS innovation project. A condition was that we clearly document our long-term plans for the project, indicating particularly the potential for wider roll-out throughout the Digital Media/Media Arts curriculum. The clear lesson is that implementing novel technical frameworks within the context of a large and centralised IT infrastructure takes time and negotiation. Rather than directly resisting existing protocols, we had much better success seeking out institutionally sanctioned currents of change within the ITS system - and then aligning our initiative with these currents.

A Brief Overview of the Curriculum

Before reporting on the effectiveness of the technical framework, it is worth saying a bit more about the Computational Media curriculum. A series of lectures focuses on positioning computational media as a mode of creative practice. The lectures consider the nature and characteristics of the digital medium, trace its relation to broader traditions of technological imagination within contemporary art and examine issues of open-source collaborative practice and re-mix. The practical workshops aim to challenge students to develop rough and ready literacy in the (typically) alien field of programming and electronics, while also demanding a capacity for critical-creative perspective. There are three assessment tasks:

1. **News game**: students (working in pairs) produce a command shell input and display game which is based upon a contemporary news story and involves elements of narrative, puzzle-solving

and navigation. This task revisits the origins of computer gaming and provides an accessible introduction to programming syntax and the shell environment. The emphasis on rendering a contemporary news story in terms of a simple game enables an oblique perspectives on media events, as well as encouraging students to think beyond standard game scenarios.

2. **Code Drawing**: students work individually to produce short algorithmic 2D and 3D animations. Early conceptual art provides one point of entry - particularly the work of Sol LeWitt (as picked up by Casey Reas in his “Software Structures” exhibition), but the task is also positioned as a means of looking beneath the hood and tinkering with a simplified graphics/games engine. This a very relevant experience for students who spend much of their time interacting with visual software but who have little understanding of the underlying technological-conceptual frameworks.

3. **Interactive Installation**: working in groups of five or six, students produce an installation that takes analogue sensor input to drive a screen-based display. The aim is to suggest creative relations between software and the electronic hardware layer and enable a playful and collaborative introduction to physical computing.

I am aware that this is an ambitious curriculum for a first year subject, but the emphasis as I have said is less upon developing sophisticated technical proficiency than upon unsettling preconceptions of the digital medium and opening up points of access to the intricate, conceptually rich language and culture of computational media.

Some Preliminary Observations

So how did the project go? The subject was generally very positively received by students. However it seems important to emphasise the continuing problems. five main issues have emerged: firstly, problems of getting the system up and running consistently on a range of lab and student machines; secondly, issues of user experience linked to working with a virtual operating system; thirdly, difficulties associated with balancing accessibility and technical depth; fourthly, dilemmas of capturing and maintaining student interest; and fifthly and finally, problems of preserving an adequate dialogue between technical learning and critical-creative engagement.

1. **Teething Issues**: We had a straight forward plan for getting students up and running with the technical framework. The VMWare player application (www.vmware.com/index.html) was installed on our lab Windows machines and we distributed DVD copies of the virtual image and associated software to students. Students were then expected to copy the 4.2Gb virtual image to their own USB drives. This would enable them to launch their individual copies of the Ubuntu GNU/Linux operating system (www.ubuntu.com) on the lab machines and access the full range of packaged development software (Scala, Arduino, etc.). If they wished they could also set up the whole system at home. We included separate installers for the VMWare player, the Java JDK (www.oracle.com/technetwork/java/javase/downloads/index.html) and Scala. We also provided

step-by-step installation and running instructions for Windows, GNU/Linux and Macintosh operating systems, as well as extensive hands-on technical support. Nonetheless the installation process proved cumbersome and all manner of small technical impediments emerged. I will describe a few just to provide a sense of the issues we faced. For a start there were a number of students in each class who had Macintosh laptops. Unfortunately, while there are versions of VMWare player available for Windows and GNU/Linux operating systems, the Macintosh OSX is not supported. The practical solution was to forget about GNU/Linux in this instance and allow affected students to access the Mac BSD-Unix interface instead. We were in no position to be FLOSS purists (especially as VMWare itself is proprietary software). Another problem related to FAT formatted USB hard disks, which have a 4Gb file size limit and are unable to handle the 4.2Gb virtual image. Our initial response was to create NTFS partitions on these drives, but then we did the sensible thing and re-built the virtual image as a multi-part zip archive (and burnt another set of DVDs). There was also the problem of students who for all manner of reasons never managed to provide a USB disk in the first place or who subsequently forgot to bring their disks to particular classes. I found myself carrying around multiple copies of the virtual image on USB key drives so that I could ensure that the whole class had access to the system. Overall, it took a few frustrating weeks to get everything up and running properly. This was hardly conducive to fostering student engagement. In consultation with ITS, a proposed future solution is to provide a self-reset (stateless) copy of the virtual image on the lab machines. Students are then welcome to take a copy for personal use, but it is not an essential condition for lab-based system access.

2. **User Experience**: One of the most appealing things about GNU/Linux is its uncluttered simplicity. Bash shell style interaction seems forbidding at first but can then quickly become a very focused and streamlined means of managing interaction with computational processes. The potential to experience this counter-intuitive ergonomic dimension was very much compromised, however, by the laborious process involved in logging on to the host Windows system, opening the VMWare player from the Novell desktop, browsing for the virtual image, launching the image via a sequence of dialogue boxes, logging on to the Ubuntu operating system, switching the latter to full-screen mode and then finally, if everything was working properly, opening up the humble bash shell application. This long-winded process clearly made it awkward to communicate an overall sense of accessible creative possibility. I doubt that this problem is easily resolved. It really entails gaining support for native GNU/Linux installation. Apart from an overall sense of experiential compromise, we found that the virtual system was a bit fragile to cope with the kind of careless use that can happen in class. For example, if the user logs off from Windows without having properly shut down the virtual image then the VMWare player creates a lock file on the image. The latter takes shape as an obscurely named file which has to be manually deleted before the system can be launched again. This problem affected many students and tended to detract from their confidence in the system. By now, most of these problems have subsided, but we need to carefully consider better means for streamlining and supporting the initial space of system encounter.

3. **Accessibility/Depth:** This is always a tricky issue. How do you enable a genuine sense of computational media without extending beyond basic systems of data representation and processing to consider key structural paradigms such as functional and object-oriented programming? Moreover, how can you introduce these paradigms in clear introductory manner? I selected the Scala language because it seemed to offer a flexible, sophisticated and multi-paradigm pathway from the simplest code constructs and modes of programming towards intermediate and then advanced constructs. Students could learn basic syntax and control structures by writing single lines of code in the bash shell, then create short interactive programs in script files and finally gain an understanding of fully compiled windowed applications. This worked well for students who were willing to put in the necessary effort to read through the study materials and complete the set exercises. Many students, however, struggled. Arguably they may have benefited more from a less ambitious and more explicitly introductory choice of programming language. The two most apparent difficulties related to the lack of general introductory tutorial materials (Scala is primarily pitched as an expert level programming language) and the potentially confusing interoperability with the larger Java programming field. The choice of Scala certainly placed a considerable responsibility on me to provide appropriate introductory resources and to carefully manage the relation between technical-conceptual depth and accessibility. Despite the problems, I believe that this has been a worthwhile effort. Overall students seem to have made much greater progress than they ever seemed to make in more obviously nurturing frameworks such as Director, Flash and Processing. I think that this is partly because they have been directly challenged. The key, however, to making this kind of challenge work is to ensure that it is framed in an encouraging and supportive way. This demands a continuing work of negotiation. I doubt there are any permanent solutions.

4. **Student Interest:** It would not be hard to rank the popularity of the three items of assessment. The interactive installation proved the clear winner, with the news game a close second and the code drawing a somewhat distant third. The students responded instantly to the physicality of working with LEDs, sensors, micro-controllers and the like. Getting them enthused about the abstract- symbolic world of programming required more effort. A likely additional factor here is that the students worked in largish groups of five or six on the installation. This enabled a more engaging, social-dialogic mode of computational media production. It also modelled key features of collaborative media art practice and inspired a nice level of technical and creative ambition. The news game did something a bit similar, but at a smaller scale. The latter's major success was in integrating the learning of basic code syntax within the context of an overview of early technical gaming platforms (command line style adventure games) and a warped examination of contemporary news events. I had envisaged that the code drawing project would prove equally popular, but the level of technical complexity was probably a bit too high and the project needed more careful conceptual framing. Students were compelled to make a big step from working with simple scripts to finding their way within a sophisticated graphics API. This may have been easier to manage if the conceptual focus had been explicitly on looking under the hood of a contemporary, but very

simplified, graphics or games engine. The austere reference to early Conceptual Art (Sol LeWitt) was probably not the most helpful or sufficiently well elaborated point of conceptual entry. Nonetheless, I feel very positive about the potential to refine this exercise to become something very useful. I devoted considerable effort to developing a 2D and 3D graphics engine and API with a simplified point of programmatic interaction. Similar to Processing, students write all of their code in a single text file which accesses the rest of the system as thought it were a compiled library. The difference is that none of the actual source code has been hidden. It is all directly accessible. There is great potential here to interrogate the fundamental structures and assumptions of contemporary forms of algorithmic representation.

5. **Maintaining a Capacity for Critique:** A final issue, arising from many of the earlier observations, involved finding a good balance and clear points of dialogue between technical-conceptual and critical-creative enquiry. The risk was to entirely lose sight of the latter in the effort to provide adequate technical instruction and to make all the various bits of student code 'work'. My major strategy here was to maintain a very artist focused orientation in the lectures, constantly emphasizing the creative-conceptual dimension of the assessment exercises and devoting a significant period of each class to the discussion of examples, the elucidation of creative ideas and the discussion and review of student work. Yet the problem runs deeper than issues of pedagogical strategy. The problem relates in many ways to the structure of the computational medium itself, which is profoundly shaped by strategies of obfuscation, encapsulation and encryption. Finding a critical language to engage with computation – to pull back its veils - is awkward because the medium is structured in terms of layers of veiling from the graphic user interface down to the layers of machine code and the flow of electrons. The more layers one descends the easier it is to lose the capacity for critically lucid engagement. But perhaps this observation, the recognition of certain level of blindness, of a certain incapacity to properly speak or gain a coherent sense of perspective, is itself revealing. If nothing else, perhaps, it justifies the need for efforts a critically informed digital literacy.

Conclusion

This paper has described the design and implementation of a FLOSS technical infrastructure for the teaching of an introductory computational media art subject within the Faculty of Creative Arts at the University of Wollongong. The project has proved a major success in terms of negotiating institutional support within the centralised University of Wollongong IT system. There remain a number of rough edges and minor compromises, but the system provides an effective model for encouraging critical-experimental interaction with underlying aspects of digital media. It enables students to delve beneath the conventional space of proprietary software tools to examine the texture, the cultural politics and the creative possibilities of underlying computational processes.

References

Fuller, M. (2003). Behind *the Blip: Essays on the Culture of Software*. Brooklyn, New York, Autonomedia.

Kay, A. and Goldberg, A. (2003). 'Personal Dynamic Media', in Wardrip-Fruin, N. and Montfort, N. (eds). *The New Media Reader*. Cambridge, Massachusetts, The MIT Press, pp. 393-404. First published in 1977.

Manovich, L. (2001). *The Language of New Media*. Cambridge Massachusetts, MIT Press.