Knowledge, Knowing and Being: an investigation of software art as a vehicle for the exploration of emerging concepts in language and cognition

by

William Hart, B.Sc. (Hons)

Submitted in fulfilment of the requirements for the Degree of Doctor of Philosophy

University of Tasmania July, 2008
Statement of originality

This thesis contains no material that has been accepted for a degree or diploma by the University or any other institution. To the best of my knowledge and belief, it incorporates no material previously published or written by another person except where due acknowledgement is made in the text.

William Hart
Statement of authority of access to copying

This thesis may be reproduced, archived and communicated in any material form in whole or in part by the University of Tasmania or its agents, and may be made available for loan and copying in accordance with the Copyright Act 1968.

William Hart
Abstract

The development of the computer has had a major impact on almost all aspects of knowledge and culture: it has had a profound impact on our understanding of complexity and intelligence; it is a mirror made from what we thought we knew about ourselves, often defining us by revealing what we are not.

Artists have been exploring the computer as a medium for over forty years, yet for most of that time art constructed through the medium of programming, has oscillated between formalism and techno-utopianism. The practise of software art is based in formal language: the manipulation of symbols and logic. The expression of interior thought in a distinctive form is often confounded in this medium.

This project considers the metaphorical representation of cognitive process (the codes of knowledge, the processes of knowing, and the states of being) in the mirror of computation and in the light of insights and theories about language and cognition that have emerged from our interaction with the universal machine. An open-ended methodology for software poiesis has been developed through making the works that form this thesis.

The thesis consists of four works, constructed from software and manifested on flat panel displays as temporal images. The Conditions for Ambient Cognition has four parts: Faith in Reason, Ontological Drift, Dialectic Seepage, and Transcendental Jitters. The four parts follow processes of cognition and being; perception, sensation, communication, action. The second work, Stories about You, is a meditation on the construction of personal and social consciousness; Communal Sense considers the communication of knowledge and development of understanding. The fourth work, The Transient Taxonomies of Art, is an examination of the concept of an ecology of ideas.

The significant outcomes of this project demonstrate the application of the methodological process, and contribute towards the development of a fluid, expressive and unrestricted form of practice in software art. This methodology is independent of a particular set of software, but does suggest a process for the evaluation of software tools and environments so that it can continue to be applied as software develops. A related aspect is the consideration and selection of the material qualities of software, algorithms and numerical function, that can be used to enhance the work and give it a
Abstract

distinctive quality. The concept of open-endedness that has emerged from the considerations of the materials of software art is crucial to resolving the confusion between algorithmic behaviour and novelty in software art.

This research has shown that with a greater understanding of the nature of complexity, and the semantics of expression through formal language, the medium can develop into a mature and distinctive form. Potentially, software art can exert a wider influence on our understanding of ourselves, our coexistence with the computer, its future application and development.
Acknowledgements

This thesis is dedicated to my Mother and Father, Judith and John Hart, who have always let their children choose their own paths through life, no matter how unlikely their prospects seemed.

Special thanks to Mary and Morgan for their insight, consideration and tolerance, particularly over the last year. Special thanks also to my supervisors, Professor Geoff Parr for his long term mentorship and bouts of mental judo; to Professor Jonathan Holmes for his generosity of spirit and attention to style; and to Professor Ross Gibson for the fruitful discussions and his objective eye.

Thanks to the people who I will never meet who have helped in this project, Paul Melis for letting me use Pyrund, Joseph Winston for his help with PyOSG, Nicolas Brodu for assistance with Crogai and Ulrich Koethe for patient help in compiling Vigra. Any piece of software stands on the shoulders of the work that has gone before, particularly open source software, for which thousands have often anonymously donated their time and expertise. This project would not have been possible without them.

Lastly, thanks to all my colleagues for their support and in particular Aaron Horsley for always helping when he could, Phillip Blacklow for his expertise with the physical world, Ona Kaukenas for proof reading and Martin Walch for being there to chew the fat.
# Table of Contents

Chapter 1 - Introduction ➤ 1

A wishful self-portrait ➤ 1
Artificial Intelligence, Analogy and Embodiment ➤ 6
The Occult Word ➤ 10
Formal and Natural Language ➤ 12
Painting and Software Expression ➤ 14

Chapter 2 - Describing the work ➤ 17

1) The Conditions for Ambient Cognition ➤ 19
   1a) Ontological Drift ➤ 20
   1b) Transcendental Jitters ➤ 21
   1c) Faith in Reason ➤ 22
   1d) Dialectic Seepage ➤ 23
2) The Transient Taxonomies of Art ➤ 25
3) Communal Sense ➤ 27
4) Stories about You ➤ 28
5) Website http://www.billhart.id.au/phd/about ➤ 29

Chapter 3 - Locating the Practise ➤ 30

Confusing Algorithms with Semantics can lead to Formalism ➤ 35
Is Software Art synonymous with Conceptual Art? ➤ 39
Software Art and Semantics ➤ 44
A Digital Aesthetic ➤ 45
The Practice of some Artists working with Software ➤ 49
   Roman Verostko and the Algorists ➤ 49
   David Rokeby ➤ 51
   Harold Cohen ➤ 54
Summary ➤ 57

Chapter 4 - Project Development ➤ 60

Prologue to the Project – Static to Animated Images 1996 - 2002 ➤ 60
General Methodology ➤ 63
2003 – Exploring the Possibilities ➤ 66
   Memetic Variations – 2003 ➤ 66
   Jitter – 2003 ➤ 73
   Moebius – 2003 ➤ 75
   Nodelands – 2003 ➤ 76
   Summary – 2003 ➤ 79
2004 – Assembling the Means ➤ 79
   Expressive Programming Environments – 2004 ➤ 79
   NPR stands for Non Photorealistic Rendering – 2004 ➤ 80
   Summary – 2004 ➤ 82
2005 – Implementing the Methodology ➤ 83
## Table of Contents

Ocular Coincidences – 2005 ................................................................. 83
Stories about You – 2005 .................................................................. 86
Summary – 2005 .............................................................................. 91

2006 – Putting it into Practice ......................................................... 92
Scene Graphs – 2006 ....................................................................... 92
Quote (Dialectic Seeage) – 2006 ................................................... 95
Jitter (Transcendental Jitters) – 2006 ........................................ 95
Classic (Faith in Reason) – 2006 .................................................. 97
Exploration Readings – 2006 ......................................................... 98
Stories about You – 2006 ............................................................. 100
Words of Art (The Transient Taxonomies of Art) – 2006 ........ 101
Summary – 2006 ............................................................................. 104

2007 – Reflection and Refinement .................................................. 105
Words of Art – 2007 ...................................................................... 105
Stories about You – 2007 ............................................................. 106
Flow (Communal Sense) – 2007 .................................................... 107
Summary - Tweaking and Resolution – 2007 ......................... 109
Words of Art (The Transient Taxonomies of Art) – 2008 .... 110
Trial Installation – 2008 ............................................................... 111
Summary ....................................................................................... 112

Conclusions .................................................................................. 114

Chapter 5 – The Materials of Software Art ..................................... 115
Programming Language as a Material ......................................... 115
Expressive Programming ............................................................... 116
Choosing A Language ................................................................... 118
Levels of Software Abstraction ................................................... 122
Drawing in 3DDimensions ............................................................. 124
The Materiality of Numbers ........................................................ 126
Scripts and Scores and Experiential Flow ................................. 126
Sampling and Synthesis ............................................................... 127
The Quality of Randomness ........................................................ 128

Chapter 6 - Conclusion ................................................................. 133

Appendix 1) Software Used in Artworks and Website .................... 137

Appendix 2) Algorithms .............................................................. 138
Spring Simulation .......................................................................... 138
Brownian Motion Simulation ....................................................... 140
Drawing Algorithm #1 ................................................................. 141
Drawing Algorithm #2 ................................................................. 142
Drawing Algorithm #3 ................................................................. 143
Drawing Algorithm #4 ................................................................. 143
Drawing Algorithm #5 ................................................................. 144
Drawing Algorithm #6 ................................................................. 145
Drawing Algorithm #7 ................................................................. 146
Appendix 3) Comparison of Different Programming Environments and Toolkits

1) Comparison of Characteristics for Expressive Programming Environments. ................................................................. 147
2) Summary of 3D software packages evaluated ................................................................. 150

Bibliography ................................................................................................................................. 151
Cited Works .................................................................................................................................. 151
Works not Cited ............................................................................................................................... 159
List of Illustrations

Illustrations, where unattributed are the work of the author.

Image 1.1 Ontological Drift, animation still, 2008 .......................................................... 20
Image 1.2 Transcendental Jitters, animation still, 2008 ....................................................... 21
Image 1.3 Faith in Reason, animation still, 2008 ................................................................. 22
Image 1.4 Dialectic Seepage, animation still, 2008 ............................................................... 25
Image 1.5 The Transient Taxonomies of Art, animation still, 2008 ....................................... 26
Image 1.6 Communal Sense, animation still, 2007 .............................................................. 27
Image 1.7 Stories about You, animation still, 2008 .............................................................. 28
Image 3.1 Jodi.org, Max Payne Cheats Only, web site, 2006 ............................................... 31
Image 3.2 Geoffrey Shaw, Legible City, photographic still, 1988 .......................................... 32
Image 3.3 Jonathan Harris, Sep Kamvar, We Feel Fine, web application, 2006 ................. 33
Image 3.4 A. Michael Noll, Gaussian Quadratic, 1962 ......................................................... 34
Image 3.5 compilation of letter forms from Alphabet: An Exhibition of Hand-Drawn Lettering and Experimental Typography, The Art Institute of Seattle, 2008 .......... 36
Image 3.6 Bill Seaman, The Exquisite Mechanism of Shivers, interactive installation, 1991 ................................................................. 37
Image 3.7 David April, Fractal Effervescence, digital image, 2006 ........................................ 37
Image 3.8 Casey Reas, Microimage, digital print on plastic, 5 x 2.8m, 2003 ....................... 38
Image 3.9 Jared Tarbell, processing applet, 2004 ................................................................. 41
Image 3.10 Robert Hodgin, processing applet, 2004 ............................................................ 41
Image 3.11 William Ngan, processing applet, 2004 ............................................................. 41
Image 3.12 LaMonte Young, Composition 1960 #10, An Anthology of Change Operations, 1963 ................................................................. 42
Image 3.13 Walter di Maria, Mile Long Drawing, Mojave Desert, 1968 ...................... 42
Image 3.14 D.X. Jones, LineFollow2D, processing applet, 2008 ........................................ 42
Image 3.15 Orel, Line in 1D, javascript code, 2008 ............................................................. 42
Image 3.16 Robert A, Kirsch, first scanned image, 176x176 pixels, SEAC, 1956.............. 46
Image 3.17 Roman Verostko, Pathway Series, algorithmic pen and ink drawing, 61x102cm, 1989 ................................................................. 49
Image 3.18 Roman Verostko, HP pen plotter using brushes, 1987 ...................................... 50
Image 3.19 David Rokeby, Giver of Names, installation photos, Kiasma, Helsinki, 2000 ................................................................. 52
Image 3.20 David Rokeby, n-ch(a(n)l, installation photographs, Banff Centre for the Arts, 2001 ................................................................. 53
Image 3.21 Harold Cohen, No. 4, lithograph and photo-montage, 59 x 59 cm, 1967 ........... 54
Image 3.22 Harold Cohen, Computer Drawing, 1974 ....................................................... 55
Image 3.23 Aaron installation photo, Documenta 6, Kassel, 1977 .................................... 55
Image 3.24 Harold Cohen, Detail from Aaron drawing, SFMOMA, 1979 ......................... 55
List of Illustrations

Image 3.25 Harold Cohen, Liberty and Friends, Ink and dye on paper, 56x76cm, 1985. .................................................. 56
Image 3.26 Harold Cohen, Aaron, with Decorative Panel, Oil on canvas, 183x137cm, 1992 ................................................... 56
Image 3.27 Harold Cohen Aaronswood #14, Digital Painting, 120x94cm, 2007 ........ 56
Image 4.1 I think therefore I am not sure..., digital print, 76x125 cm, 2002. ........ 62
Image 4.2 I think therefore I am not sure... (detail) ............................................. 62
Image 4.3 Words, animation still, 2002. ............................................................... 62
Image 4.4 The Writing Machine, C++ program, 2002. .......................................... 63
Image 4.5 Guillaume Apollinaire, La Mandoline l'oeil et le Bambou, from Calligrammes, 1912-18 ........................................... 68
Image 4.6 Evan Roth and Mark Asare, Baskerville Old Face, Flash animation, 2003 68
Image 4.7 Unattributed image produced with Hansol Huh, Typedrawing, digital image, 2006 .................................................. 68
Image 4.8 source images for Memetic Variations, 2003. .................................... 69
Image 4.9 Memetic Variations, screen capture, 2003 ........................................... 70
Image 4.10 Journal sketches for jitter, 2003 .......................................................... 72
Image 4.11 jitter, C++ program, 2003 ................................................................. 74
Image 4.12 Moebius, animation still, 2003 ............................................................ 74
Image 4.13 Visual Thesaurus, Thinkmap Inc, Java applet, 2008. .......................... 75
Image 4.15 Ben Fry, Valance, C++ application, 1999 ......................................... 76
Image 4.16 Jared Tarbell, PondCode, Flash interactive, 2001. ............................ 76
Image 4.17 Nodelands, Flash interactive, 2003 ..................................................... 76
Image 4.18 Journal entry for expressive programming, 2004. ............................. 79
Image 4.20 Journal entry, spring dynamics, 2005. ............................................... 83
Image 4.21 Ocular Coincidences, animation still, 2005. ..................................... 85
Image 4.23 Jon Thompson and Alison Craighead, Decorative Newsfeeds, video projection, 2004. ........................................ 87
Image 4.26 Journal entry, Python ⇒ C++, 2005. .................................................. 88
Image 4.25 Ed Burton, Sodaplay, Java applet, 2004 ............................................. 88
Image 4.27 Stories about you, animation still, 2005. ............................................. 89
Image 4.28 Journal entry, Memetic Variations components, 2006 ........................ 92
Image 4.29 Memetic Variations (straight text), animation still, 2006 .................... 93
Image 4.30 Memetic Variations (curved text), animation still, 2006 ...................... 93
Image 4.31 Quote, animation still, 2006 .............................................................. 94
Image 4.32 Camera stand, 2006. .................................................................. 95
Image 4.33 jitter, animation still, 2006 ............................................................... 96
List of Illustrations

Image 4.34 Source images for Classic, 2008............................................................. 96
Image 4.35 Classic, animation still, 2006................................................................. 97
Image 4.36 Ken Perlin, Kinetic Poetry, Java applet, 2001....................................... 98
Image 4.37 Noah Wardrip-Fruin et. al. Screen, virtual reality cave, 2002-2005..... 98
Image 4.38 Exploration Readings, animation still, 2006....................................... 98
Image 4.39 Stories about you, animation still, 2006............................................... 99
Image 4.40 KingHost Screensaver 1.0, application.................................................. 101
Image 4.41 Karl Sims, Evolved Virtual Creatures, animation still, 1994................. 101
Image 4.42 Jon McCormack, Eden, installation photo, 2004.................................. 101
Image 4.43 Tom De Smedt and Frederik De Bleser, Prism, digital image, 2005... 102
Image 4.44 Words of Art, screen capture, 2006...................................................... 103
Image 1.1 Ontological Drift, animation still, 2008.................................................... 103
Image 4.45 Words of Art (The Transient Taxonomies of Art), animation still, 2007... 105
Image 4.46 Stories about You, animation still, 2007............................................... 105
Image 4.47 Edge reduction and drawing process for Stories about You, 2007..... 106
Image 4.48 Flow (Communal Sense), animation still, 2007.................................... 108
Image 4.49 Christopher Dodds, Adam Nash and Justin Clemens, Babelswarm, screen capture from Second Life, 2008................................................................. 109
Image 4.50 Gallery installation planning; 3D visualisation, Monitor frame design and prototype, 2008................................................................. 111
Chapter 1 - Introduction

This chapter introduces the philosophical and theoretical background to the project; contemporary ideas in linguistics, cognition and philosophy, our relationship with the computer and the expressiveness of programming are considered.

In Chapter 2 the exhibition-based component of the thesis is described and summarised. Chapter 3 contextualises and locates the thesis within the broader field of historical and contemporary art and technology, with specific reference to the research-based practices of Harold Cohen and David Rokeby. The conceptual content of the work is considered in the broader context of twentieth century art.

Chapter 4 traces the development of the project in more detail including prior work, methodology, and all work undertaken in the course of the research. Chapter 5 presents experiential and methodological outcomes of the project and observations on the material qualities of the medium. Chapter 6 summarises the conclusions of the project.

A wishful self-portrait

The computer is a kind of wishful self-portrait... a compendium of abilities we have as humans aspired to but are not very gifted at. We need a much clearer understanding of this complex relationship. Without this understanding we will be unable to find an appropriate partnership with our creations.¹

I would argue that there have been three great human innovations in cognition: speech; drawing and writing; and the computer. The first had no dependence on technology. The second is dependent on technology (mark making tools and surfaces) and encompasses the ability to communicate independent of time and presence, but also encompasses pictorial and symbolic thought, mathematics, writing, and representation. The third, the computer, is an abstract idea of the universal machine, made manifest through the most precise application of craft and technology we have ever known. The universal machine is a chameleon, a simulation device, but also a

mirror that reflects back our belief about ourselves, and therein lies the essence of Rokeby’s challenge. At this time, we are infatuated, locked in a half fearful, half ecstatic embrace with this technology, undergoing a personal and social metamorphosis the outcome of which is difficult to predict.

As speech and language gave us the ability to communicate our inner states of mind, and must have led to the internal theory of mind that enables us (sometimes) to think and feel as others, to know their mind, so did drawing and writing allow the compilation and codification of experience and knowledge. I argue that the computer enables us to reflect not just on our abilities to accumulate knowledge, or to develop a greater understanding on the processes of knowing, but also to illuminate aspects of the nature of our being. The title of this thesis, Knowledge, Knowing and Being, encapsulates the nexus that exists between the transformation of knowledge, seen as an objective collection of codes, its perception and understanding, and how these relate to our existence. This transformation through the senses of knowledge into understanding is cognition. Knowledge can be objectified, but knowing is emotional, and the feelings that come from this form of our being. This is my interpretation of psychologist / philosopher Nicholas Humphrey’s theory of consciousness, which like much recent thought on mind and consciousness, rehabilitates emotion and sensation as key elements of cognition. Humphrey considers art as analogous to the sensation of consciousness,

What’s it like to be a painting? What a question. The answer has to be: probably not much. But the point of asking such a question is to prompt the thought that perhaps certain works of art do have the property of being in a special way “like something,” a something that again is very hard to capture in words.

Allow me a minute to play with just this thought. Suppose there is an analogy between a work of art and “a work of sensation,” where can we go with it? Well, to start with, we might want to use artistic methods and media instead of ordinary language as analytic tools for exploring the nature of phenomenal experience.

Through its developmental history the computer so far has illuminated two key intellectual and philosophical ideas. The first, through discrete mathematics and the prodigious capacity of the computer to iterate, is an emerging understanding of

---

Introduction

complexity. Since the enlightenment the key methodological assumption has been that apparently complex systems can be understood through studying the behaviour of individual components of that system: reductionism. The computer as an instrument for approximating and exploring complex systems has demonstrated that the behaviour of a system is not always predictable through an understanding of its component parts. The whole can be more than the sum of the parts: a gestalt. The process by which complex properties or behaviours arise from the interaction of the elements that form the gestalt is referred to as emergence. Many humanists have enthusiastically greeted the demonstration of principles of emergence through simulation as it represents a philosophical escape from the grim mechanisation of existence that the reductionist approach has so often represented.

The second key concept is emerging ideas about the nature of intelligence, consciousness and cognition that have occurred through the failure of the Strong Artificial Intelligence research program to achieve its goals. This failure to develop general-purpose problem solving systems based on rules and syntax, has led to a re-evaluation of logic and symbolic processing as the engine of the mind.

...a more or less explicit decision was made in cognitive science to leave culture, context, history and emotion out of the early work. These were recognized as important phenomena, but their inclusion made the problem of understanding cognition very complex. The "Classical" vision of cognition that emerged was built from the inside out starting with the idea that the mind was a central logic engine. From that starting point, it followed that memory could be seen as retrieval from a stored symbolic database, that problem solving was a form of logical inference, that the environment is a problem domain, and that the body was an input device. Attempts to reintegrate culture, context, and history into this model of cognition have proved very frustrating.  

From this re-evaluation and research insights that have come from neuroscience, a number of alternative approaches to language, cognition and consciousness have been emerging over the past twenty-five years. These include Integrationist Linguistics and Distributed Cognition; what they share in common is a rejection of the classical view of the symbol processing or code-based nature of language and mind, with an

---

Introduction

emphasis on meaning derived from the context in which all thought and communication takes place.\(^4\)

It is the interface between the formal symbolic languages of information processing and the ambiguous context dependence of human natural language that this project explores. The artworks that form the thesis are an exploration of the mysterious processes whereby we encapsulate and clarify our thoughts and feelings within the graphic forms of symbolic language (words, symbols, pictures), and the equally mysterious communication process whereby we transmogrify these symbols into understanding. I refer to the symbolic form as Knowledge and the perception and actualisation of knowledge as the process of Knowing. The third term in the title, Being, refers to self awareness, the totality of the self in the present, those fairly fleeting moments when we are fully conscious of ourselves and our existence; what psychologist Abraham Maslow termed the peak experience or mystic G.I. Gurdjieff called self remembering.\(^5\)\(^6\) Through visual representation, this project investigates the processes that integrate the struggle for communication with the pleasure of understanding and the physicality of awareness.

This investigation is conducted from the perspective of a visual arts practitioner, not as original research into cognitive science, philosophy, mathematics or computing, nor as an illustration of specific concepts or theories from those disciplines. Rather the synthesis of concepts and ideas from a broad field is used to contextualise and address some of the core theoretical and practical challenges facing the software artist. At the core of this project is an apparent paradox; a philosophical position that rejects the symbol processing paradigm of human cognition, but allows that that cognition may be extended through engagement and expression through formal symbolic language. It is through taking this position and examining this conundrum at the interface

\(^4\) Issue 26, 2004 of the journal *Language Sciences* has a number of articles about the relationship of these two disciplines. For example see


Introduction

between formal and natural language that this project contributes to the practise of software art.

The results of this investigation are manifest as temporal images (images that have a dimension of time) on flat panel displays. Software art is, in the context of this project, the representations expressed through programming by artists. Theories of language and cognition that inform the artworks are also intertwined with the medium of expression, so that this project is also an investigation into the modes of expression and the material properties of software and numbers; the relationships between formal (programming) and natural (human spoken or written) language, and the semantic potentials and limitations of algorithms.

Expression and the adjective expressiveness are used throughout this exegesis in a number of related senses. Firstly and formally the expressiveness of a programming language is considered in terms of; the compactness of the code; how easily it is comprehended and the diversity of concepts and constructs that it can represent. Secondly, in the more traditional art making sense, of manifesting interior cognitive states through a physical medium. While the significance of this latter concept of expression to visual art has been challenged by a diversity of 20th century art practises, it is considered here in relationship to distributed cognition. From this perspective, the artistic medium, whether paint or code, extends or enhances the cognitive abilities of the artist, resulting in a cognitive artefact: the artwork. Expressiveness in this sense refers to the interface between the cognitive systems of the body, and the cognitive extension of the art-making medium. It refers to both facility and to the potential for a diversity of outcomes, and in this way the two uses of expression are closely related.

For the viewer the work does not seek to be didactic in the sense of presenting a specific position, but aims to provoke a meditative response and, through a longer term engagement, a consideration of the relationships between self, language and image. The works are all based on the animation of discrete graphic elements in three

---

7 The relationship between software (the text-based program, assets such as images, and associated libraries and operating environment) and hardware (the physical equipment of display, graphics card, cpu etc) is complex and difficult to separate. I think of the manifestation of the work on a screen as the artefact, but the program as the medium.
dimensions although, in some works, they are displayed with a two dimensional orthographic projection. The majority of the works animate discrete elements to form imagery, using methodologies abstracted from simple drawing processes. The works employ a variety of simulated dynamical systems to create different qualities of movement. Viewing at a distance reveals the subject imagery of the animation, while closer viewing distances emphasise the movement and individual elements.

The works seek to evoke a meaningful response from the viewer, via a combination of ambiguity in the imagery, the content and form of the individual elements, and the dynamical qualities of movement, together these depict contrasting linguistic and cognitive processes. The sequencing of imagery in individual works is not orchestrated, but allows for chance conjunctions between works.

Artificial Intelligence, Analogy and Embodiment

The ability to make meaningful marks, to draw, must be one of the most significant technological achievements of humanity. The technology of surface and marker, pencil and paper is one of the most flexible and powerful of thinking tools; it is hard to imagine a means of representation more suited to enable the transition from vague sensation to abstract thought, nor one that can allow expression of ideas in such a diversity of forms from pictorial to symbolic. Drawing enables cognition that expands beyond the serial moment of one thought-image followed by another. Marks act as placeholders for ideas, the two dimensional space of paper allows many more and even contradictory thoughts can be simultaneously arranged in different geometries. Today we tend to think of drawing as the sole province of children and artists without

It is, of course, a simplification to construct a continuum of drawing between representation and abstraction, as it doesn’t allow for differences between, say, illustrative and observational drawing, diagrammatic and geometric drawing, or analytic and emotive drawing, to name but a few. However Integrational Linguistics makes no distinction between drawing and writing at all, claiming that the association of writing with speech is a category error, and that drawing and writing are part of the same group of strategies to communicate.

See Chapter 1, Sally Elizabeth Pryor, Extending Integrationist Theory through the Creation and Analysis of a Multimedia Work of Art: Postcard from Tunis, University of Western Sydney Nepean, 2003.

Introduction

considering the lists we scrawl, doodles in agendas, the crude sketchy diagrams we make to explain something or to just figure something out, as drawings.

For me, the computer has something of the qualities of drawing plus something more; as well as being the sheet of paper, it can also be the library; and as well as developing the plan, it can be the workshop where the idea can be given form and tested.  

There are lessons to be learned from any intellectual effort. In respect to achieving its stated goals, the Artificial Intelligence research project was one of the great failures of the 20th century military-industrial complex; some of the remarks researchers and commentators made about the progress and achievability of A.I. seemed to typify a kind of hubris about the explanatory power of formal symbol processing. Strong A.I. is based on the assumption that intelligence can be expressed as a set of rules for general purpose problem solving, 

An apparent corollary from the strong A.I. position is that if a machine has the ability to solve general-purpose problems, then it is displaying intelligence, and hence self-awareness will follow. Many people find the idea that consciousness can be reduced to a set of rules discomforting and diminishing. I suggest that this discomfort with the concept of strong A.I. and the then-dominant behaviourist paradigm in psychology contributed to the anti-technology movements of the 1970s. It was only

---

9 Admittedly, as yet its interface is too formal; it does not have the ability to doodle easily, to flip between sketch and symbol, to take that very vague inexpressible idea and give it unconscious reign, and then recognise it and bring it into focus.


12 Sherry Turkle, "Computational Technologies and Images of the Self., Social Research 64.3 (1997).

13 This is reflected in the archetypal portrayal made of the computer of the times, soulless (and often insane) disembodied machines which make poor ethical and moral choices, from HAL9000 in Kubrick’s 2001 A Space Odyssey, through to TV shows such as Dr Who, The Green Death, Star Trek, The Ultimate Computer. Compare this to the embodied robot that is
Introduction

with the failure of the Strong AI approach to produce results in the 1980s, (and of course the actual development of the personal computer) that the computer has come to be almost universally embraced as a cognitive tool. It is used as an aid to memory; rapid acquisition of knowledge; as a tool for facilitating expression with text, sound and image; as an avenue of social engagement and entertainment; as a means of exploring alternative courses of action. In a short period of time the computer has become so deeply embedded in the fabric of technologically developed societies, that many of us find it difficult to imagine functioning without it, without in some way being crippled or diminished.

An underlying assumption since the 1950s has been the idea that computers are capable of intelligence. This stems from the view that human intelligence is separate from being; that logic was the essence of reason; that as humans construct a machine to manipulate symbolic logic, so the machine itself must be capable of intelligence.

The curious thing about computers and software is that, even though software execution is just symbol-processing, the computer is a simulation machine of extraordinary flexibility; a universal computer can compute any problem that is computable. The question is, are intelligence and consciousness computable? Physicist David Penrose argues that consciousness is non-algorithmic, that it is not capable of being modelled on a Turing machine.\textsuperscript{14} Others would argue that with the development of complexity theory, and understanding it as a counter to reductionism, that some systems of rule-based logic can give rise to behaviours that cannot be described or understood in terms of the system that implements them. This insight has led to the foundation of the A-life research project. In a similar fashion to strong A.I., A-life posits that life can be synthesised in a computer and that from a sufficiently sophisticated and powerful artificial ecosystem life, and eventually intelligence will emerge. Physicist David Deutsch argues that the fundamental structure of the universe is quantum computing and that we (and all we perceive)

Introduction

could be a simulation run by an advanced intelligent species undertaking research or play near the end of the universe.\textsuperscript{15} It has become a common trope of recent cyberpunk fiction to speculate that, if the electrical and chemical activity of the human brain could be read in sufficient detail, it could be transferred to a simulated brain, which would then be capable of self-reflection. Perhaps the future will reveal one or all of these ideas as quaint, just as we now see the notion of the disembodied intelligent machine of logic as the result of a fevered cold war imagination.

The failure of A.I. historically to solve apparently easy problems of biological systems such as language translation or vision, has led to a growing realisation of the importance of embodiment; that the way we think is connected to the way we perceive and interact with the world, that our reason is inextricably tied to our emotions and our bodies, that there is an intimate linkage between being and knowing. Reason is not manifest through processes of inductive or deductive logic, but occurs fundamentally by a process of analogy, of finding like within difference, a connection to a familiar or understood. The developer of the embodied mind thesis, cognitive linguist George Lakoff, argues that abstract reasoning depends upon the emotions, physical movement and the sensations of the body.\textsuperscript{16} Neurologist Antonio Damasio has arrived at a similar position with his Somatic-Marker hypothesis, by studying patients with selective brain damage to the frontal lobes of the brain, and concluding that emotion and feeling are essential to goal-setting and decision-making.\textsuperscript{17,18}

Visual art is the process of thinking through making, or understanding the world via creating representations. It is, in essence, a process of constructing analogy.


\textsuperscript{18} There is something profoundly disturbing and sad to read descriptions of people with different types of neurological damage, and the realisation of what a fragile thing the mind and self is. I have found no greater argument for the material basis of mind.
Introduction

Considered in the light of these emerging understandings of cognition, Visual art is not inherently inferior to other cognitive processes for gaining and communicating knowledge. Art historian and theorist Barbara Stafford argues, in Visual analogy: Consciousness as the Art of Connecting, for a rehabilitation of analogous reasoning from the dominant allegorical mode of postmodernism,

*I want to recuperate analogy, then, as a general theory of artful invention and as a practice of intermedia communication. Knowledge is a heuristic system always in pursuit of equivalences for one thing or another. It results when abstractions are made concrete, when family ties between distant or separated events are exposed.*

With these insights in mind, it is through the medium of programming as a process of constructing and expressing analogy, for making connections and associations, for which the computer can embody a similar cognitive role to paper and stylus.

The Occult Word

Following on from drawing, the development of first spoken language and subsequently written language can be argued to be among humanity’s most powerful technological achievements. Symbolisation, the process by which images become symbols, is so effective that we are tempted to think of it as transparent. Consider letterforms; for us they no longer have a pictorial significance; we are able to almost instantly recognise a vast variety of variations of the same letterform as that symbol. We have become so adept that it is tempting to conclude that the ability is innate. We are trained in the ability to recognise particular shapes and forms from a very young age, while our neurons and synapses are still plastic; this is an adaptation (in Darwinian terms an exaptation) of our innate abilities to extract patterns from our visual senses:

*…one wonders just how many potential synaptic connections were lost to members of the civilized world when their ancestors abandoned a foraging lifestyle and settled in villages, towns, and cities. Anyone who has seen an African tracker scan a featureless plain and locate a distant pride of lions, invisible to everyone else in the car, will appreciate the impoverished nature of the synapses in his or her own visual cortex.*


Introduction

Rather than impoverished, the same abilities are still developed but are now used differently to effortlessly discern the shapes and forms of symbols, and they are continuing to evolve and adapt under the pressure of increasingly complex graphical representations.

Historically the power of *naming*, of being able to abstract and label a concept and so render it symbolic, has been a source of fascination. For instance the epic of Gilgamesh (the first recorded story), tells of a complex character, simultaneously a wise leader and ruthless tyrant, part man and part god. In the story, Gilgamesh had mysterious hidden knowledge of the true names of things recovered from earlier times and thus secured his immortality by carving his story on stone.²¹

Modern fantasy writing is littered with references to those who gain power over others and the natural world via the knowledge of true names or occult languages. The hermetic maxim of ‘as above so below’ is the essence of this magical knowledge; if you find a way to abstract an aspect of the physical world to the transcendent sphere of ideas, then manipulation of the idea gives control over the physical world.²²

Florian Cramer, in *Words made Flesh*, surveys and analyses the history of occult systems of knowledge in the context of generative processes and connects them with 20th century explorations and developments of generative practice and computer code, and concludes by defining the concept of software in a cultural context.²³ The jargon of software is steeped in the language of mysticism and the occult; there are software wizards, gurus who have special knowledge, software daemons that undertake hidden tasks, and many software packages and tools have magic in the name. The language of code is hidden from the pedestrian user; one must become an adept to learn the secret names and language that give the power to control the

---


²² Translated by Dennis W. Hauck the full form is "That which is Below corresponds to that which is Above, and that which is Above corresponds to that which is Below, to accomplish the miracle of the One Thing."


Introduction

machine beyond the capabilities of normal users. It is not just a measure of ability or expression, but also a precise knowledge of incantation, of knowing the right name and how to invoke it, or being able to access the arcane volumes of information to find the right command. This is a very different cognitive skill from that of communicating abstract and fuzzy meanings to others through written and verbal language.

The computer became one of the dominant metaphors of the twentieth century and this remains the case. With the failure of the strong A.I. project, almost all intellectuals would reject the direct analogy of the brain to the computer, but it is almost impossible to discuss cognition without talking about networks and processing. For instance, a recent public project to develop awareness and resources for literacy, *Children of the Code*, bluntly equates written expression with software;

> The "code", the technology of written language, is the most influential invention in the history of history. It is the "OS" (operating system) of civilization.\(^{24}\)

As with the cautionary tales of wizards who misuse their power, metaphors can be dangerous things; they can grant power, but at the same time blind us to other possibilities. We have become so fluent at reading that it seems as though the translation of image to symbol to meaning is transparent, forgetting that, like the hunter on the savannah who can effortlessly read the terrain, we are looking at images. It is our training to read them as symbols; the associations built up with particular shapes and forms give them their specific meaning. This is not the same as the way a computer processes symbols.

**Formal and Natural Language**

The power of language natural and formal, visual, textual and numeric is to abstract concepts from the specific and particular to the universal. The *naming* of objects, concepts and actions is a powerful archetypal action. *'As above, so below'* is a kōan describing the relationship between naming and power, a characteristic exhibited by most systems of organised knowledge (and other hierarchies). For example: the taxonomy of botany and zoology, the lexicon of art and artists of art history, the jargon of a bureaucracy.

Introduction

The relationship between meaning and form in natural languages is more ambiguous and even more so in many visual representations. Natural language is not static, it evolves; the meanings of a word change with usage, new words are coined to describe a new experience, concept or object. Some become antiquated, obsolete or are made irrelevant by new insights.

Some word/concepts and syntactical structures are so powerful that they can impede other ways of seeing and understanding. Notions of self, consciousness, and causality, for instance, deeply colour our thinking and dialogue about the nature of existence. Endless debates about mind/body dualities, objectivity/subjectivity and other apparent paradoxes are born from the limitations of the entrenched conceptual and perceptual framework, and the syntactical structure of language, rather than external fundamental truths.

In a formal language, the structure or syntax determines what can be said. Programming is the process of constructing complex abstractions from simpler syntactical forms to create function. General programming languages are said to be Turing-equivalent if they can compute every function that a theoretical Turing machine can; in other words, in theory, everything that can be computed in one programming language can also be computed in another. However, a concept in one programming language may be very difficult or lengthy to express as a code construct; in another, it may be compact and straightforward. In practise the range and

---

25 In linguistic theory this is expressed as the Sapir – Whorf hypothesis; in its strong form the hypothesis implies that language determines what we can think. More generally accepted are weaker versions of the hypothesis based in relativity – ‘people who speak different languages perceive and think about the world quite differently’.


26 Quantum physicist, David Bohm, aware of the limitations that the emphasis on nouns in English places on our understanding of causality and process, developed a language called the *rheomode* that he forced his graduate students to use. See Chapter 2, “The rheomode – an experiment with language and thought” in


27 Unlike natural languages, a statement that is not syntactically correct has no meaning – It does not compute!

28 The Sapir – Whorf hypothesis has a more deterministic application to programming languages than to natural language.
Introduction

character of expression available to the programmer is dependent on which
programming language is used.

There are a large number of programming languages that have been developed over
the last 50 years (around 8500), of which nearly all are have English key words and
were developed by engineers and computer scientists. The ten most popular
languages account for around 95% of all programming activity. There are specialised
programming environments that have been developed for artists, built around core
functionality, an animation or sound-processing engine for instance. The choice of
language for artist-programmers is often dictated by the capabilities they require,
rather than by the syntactical structure of the language. A different approach has
been taken with this project; rather than choose an authoring environment for its
capabilities, a programming language has been selected for its expressiveness, and
then extended with the desired capabilities. A discussion and description of this
methodology is contained in Chapter 4 and 5.

Painting and Software Expression

Visual images, as subjective representations, have the potential to be able to question
the relevance of a worldview or conceptual framework in ways in which an analytical
approach based in language cannot. A visual representation can give form to an
intuition or a hunch; make explicit the conceptual tensions that exist in any state of
being. To return to Nicholas Humphrey’s quote ‘certain works of art do have the
property of being in a special way “like something”, a something that again is very
hard to capture in words.’ We know what Humphrey means but, like him, find it

David Cerezo, On the Sapir-Whorf Hypothesis and Its Relation to Programming Languages, 2004,

Diarmuid Pigott, Hop: An Interactive Roster of Programming Languages, 2006, Available:
http://hopl.murdoch.edu.au/

Anonymous, Tiobe Programming Community Index, 2008, Available:
http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

For instance the Actionscript language used with Adobe Flash has inbuilt functionality for
manipulating and displaying 2D geometry, which can be viewed in most web browsers, but is
not as suited for live performance as the signal processing engine of MAX/MSP is.

Humphrey, Seeing Red: A Study in Consciousness. p112.
Introduction

difficult to describe. The difficult question then for software art, and for our relationship with our ‘wishful self-portrait’ the computer is, to paraphrase Humphrey's, under what circumstances can the products of expression through a formal language (i.e. a piece of software art) have that property of being like a sensation of phenomenal experience, of consciousness. The answer, I believe, is one that can only be arrived at through exploration and experiment, an understanding of the relationship between the forms constructed through formal expressions, and the strength of semantic associations that those forms can support.

Consider the similarities and differences between painting and programming. Painting, although we tend to forget, is a technologically mediated activity; the industrial technology of pigments, mediums, surfaces and brushes influence the range of expression that the artist can have. There is a significant investment of time in learning the interface before it becomes transparent, that is, developing the degree of facility with the processes of mixing and applying paint that they become unselfconscious. Occasionally there is discussion about a formal language of paint or picture making and, while formalist approaches to painting were a significant aspect of modernism, the reality is that there are a myriad approaches to the medium. It is the plasticity of painting, its open-endedness that gives it its enduring appeal. Open-endedness is a quality describing the range of distinctive possible outcomes that a process can have - distinctive in the sense of not just being different, but different in kind, different in semantic state. It is different to complexity or entropy; it is a measure of the potential for the construction of meaning.

---

33 I assume that is possible, my argument is as follows. All digital applications and tools have some level of programming (formal language) involved in their construction, so any creative act or artefact created using digital technology is based in part on a use of formal language. There are numerous examples of sound, film and video produced in the last thirty years through digital technology that are considered to be significant works of art. However these works treat the underlying technology as a transparent carrier for the semantic content of the work.

34 In any process of learning a new skill or ability, there is a stage of intense self-conscious concentration on learning the task. With repetition the need for this concentration disappears and requires little or no conscious thought.

35 This is a dreadfully clumsy word, I’ve searched for an alternative, but nothing quite seems to fit.
Introduction

Programming, like painting, requires the development of a range of skills; physical facility with keyboard and mouse, an understanding of formal logic and the conventions of the programming language and familiarity with the APIs of the software libraries being used. Programming in the practise of software art can often be characterised by poorly defined initial concepts or ideas, which are clarified through developing the project. The goal of this project is, through the complementary processes of making and reflection, to explore the conditions for an open-ended interface between human and computer, expression through software.
Chapter 2 - Describing the work

This chapter describes the works that form the final submission. A detailed discussion of the context and the development of the works are contained in the following chapters. Each work consists of four components:

• A semantic layer of digitised imagery (photographs and drawings) and/or appropriated texts. It is from this layer that the content of the work derives.

• A software animation engine, and associated scripts for preparing the data. The program acts like an armature that supports the semantic layer. It is the interaction of the software engine and the semantic layer which gives the work its meaning.

• The software environment that the animation engine requires to operate, consisting of open-source third party packages and libraries.36

• Computer hardware and display devices, which form the physical support for the work.

Internal processes for making artworks are difficult to articulate succinctly. The methodological process starts with a sense image, a non-verbal impetus that could be formed from an unresolved collision of ideas or a new insight, or a poetic inspiration. Rarely does the image have enough clarity initially; rather, over time, the image returns periodically, generally becoming more focused with fresh insights. Clarification of the sense image leads to a schematic sense of the components and capabilities the work will require, which usually requires pragmatic research to find and construct the necessary software environment. Similarly, potential semantic content for the work is acquired through photography, digital imaging and web research. The development of the animation engine is a cycle of programming and evaluation of capability, dynamics, form and content, followed by longer periods of reflection to evaluate the sense of resolution of the work. The level and type of description and analysis outlined below occurs near or after the completion of the work.

36 See Appendix 1: Software Used in Artworks and Website
Describing the Work

The thesis consists of this exegesis, four artworks and a web site:

1. *The Conditions for Ambient Cognition* is a meditation on the relationships between knowledge, knowing and being. It consists of four component works, each displayed on a single LCD panel:
   a. *Faith in Reason*
   b. *Ontological Drift*
   c. *Dialectic Seepage*
   d. *Transcendental Jitters*

2. *The Transient Taxonomies of Art* is displayed across three LCD panels and is a simulation of an ontological ecosystem.

3. *Communal Sense* a meditation on the reconstitution of coded insight between generations.

4. *Stories about You* explores the construction of personal and social conscious identity.

5. The website [http://www.billhart.id.au/phd/about](http://www.billhart.id.au/phd/about) contains documentation of experiments, developmental stages and methods of this project.

An initial impetus for this project was an interest in the potential of LCD flat panel displays as a transparent medium for the display of digital imagery. Mounting the screens on the wall, hiding the cables and removing the plastic surrounds displaces the screens from their technological consumerist or corporate context, inviting a reading of the work as pictures. The appearance of the imagery is closer to the transmissive imagery of a light box than imagery formed through reflected light, such as a painting or drawing. The degree of detail inherent in a LCD panel invites intimacy and inspection in a way that is difficult to achieve with the projected imagery. In a gallery space the viewer can experience a wide range of scale relationships to the work. Up close, individual elements can be discerned and read; from further away dynamic patterns become apparent and from further still, in some works, images formed through the process of animation emerge.

---

37 In 2006 I wrote a paper *The electronic image – object to picture*, which explored the technological trajectory of the display, physical depth of the display against resolution and cost over time, and the likely influence of this on the uses and relationship to moving imagery. See, [http://toolshed.arthbt.utas.edu.au/billhart/papers/bite](http://toolshed.arthbt.utas.edu.au/billhart/papers/bite)
Describing the Work

1) The Conditions for Ambient Cognition

Ambient is an adjective relating to the immediate surroundings of something, and cognition is the mental process of acquiring knowledge and understanding, so the work is exploring the circumstances that surround thought and understanding. It consists of four parts: Faith in Reason, Ontological Drift, Dialectic Seepage, and Transcendental Jitters. The four works follow processes of cognition and being: perception, sensation, communication, and action.

The gestation of this work has been long and complex (see chapter 4) and it is through the development of the work and associated reflection and reading that I have arrived at an understanding of language, cognition and the relationship to the computer best described by the collection of theories known as Distributed Cognition. Distributed Cognition is a relatively recent offshoot of mainstream cognitive science that can be traced back to the 1980s. It is described as being able to,

…bundle together some coalescing recent lines of research in philosophy of mind, developmental psychology, robotics, cognitive archaeology and anthropology, dynamical systems approaches to cognition, and human–computer interaction.

There is no unified approach to distributed cognition but, broadly, a number of characteristics can be identified:

- Cognitive processes are not necessarily contained within the brain or body of an individual; they may be distributed across a social group or amongst artefacts.
- Language and communication is an activity based in the participants’ context of current and prior experience, not a transmission of symbolic codes from one mind to another.
- Similarly, cognition is more than symbol processing; it is an embodied activity, dependent upon context. While the nature of mental representation is

---

38 Hutchins, Smelser and Baltes, "Distributed Cognition". p2068.
39 Sutton, "Representation, Levels, and Context in Integrational Linguistics and Distributed Cognition". p505.
40 Hutchins, Smelser and Baltes, "Distributed Cognition". p2068.
41 Sutton, "Representation, Levels, and Context in Integrational Linguistics and Distributed Cognition". p509.
Describing the Work

...disputed, it is considered more complex than the internal translation of perceptions to a symbolic form.

1a) Ontological Drift

Ontology is the branch of metaphysics that explores the nature of being. In computer science it is the representation of a set of concepts and their relationships with a specific area of interest or field of study: a domain. There is a tension between the two interpretations of the terms; one formal and precise, the other reflective and fuzzy. Drifting is to move involuntarily into a certain situation or condition. In Ontological Drift words are the actors, driven from one context to another.

The words come from two Wikipedia lists of words, a list of words ending in -ism (3487) and words ending in -ology (804). Words ending in -ism relate to several categories of usage, often slightly derogatory. They relate to an ideology, denoting a subjective belief or doctrine, but can also refer to pathological conditions. Words ending in -ology denote a branch of knowledge; there are many more -isms than -ologies in use in the English language.

So the text of the work contains an inventory of labels for subjective practise and objective study: belief and rationality.

The words are rendered on flexible arcs (defined by three anchor points) in a three dimensional space with an orthographic viewpoint. The animation system uses a simulation of a spring attached between each anchor point and a corresponding key.

---


43 These lists have since been deleted but similar lists are available at http://en.wiktionary.org
Describing the Work

point. The spring strengths are quantised to five values to give some cohesion to the movement and, over time, the spring strength and damping coefficients increase to damp the oscillations.\(^4^4\)

The anchor points are derived from source imagery using Drawing Algorithm #6.\(^4^5\) The source images are from a personal collection of incidental photographs of objects, family members, travel photos, the local neighbourhood. In general, the images have been chosen for their distinct form or structure; some have been modified to remove the background so that the precise form of the image will render clearly. The images are intended to convey a subjective sense of interior imagery - a glimpse of a memory or imagining that may be occurring as you talk to another, rather than objective documentation of experience.

The gestation of this work has been long and complex.\(^4^6\) The intention is to probe the connection between the contextual soup of knowledge and belief, the resonances of association through which we swim, and the direct sensory experience we have of the world.

1b) **Transcendental Jitters**

This work makes a connection between the detritus of the body, the stuff (skin, hair, bacteria) that we constantly shed and exchange with our environment, where we exist as a mobile collaborative colony of cells, and the abstracted world of sensation where we perceive ourselves as one. The dynamics engine simulates

\(^4^4\) See Appendix 2: Algorithms: Spring Simulation

\(^4^5\) See Appendix 2: Algorithms: Drawing Algorithm #6

\(^4^6\) See Chapter 4: *Memetic Variations* for a discussion
Describing the Work

*Brownian* motion (the jittery motion of microscopic particles in fluid) to animate nail clippings on a two dimensional plane. The energy of the nail clippings are affected by the tonal value of an invisible key image. Where the image is dark, the clippings lose energy and move more slowly; this causes the clippings to cluster over the dark areas of the key image, rendering the image. The key images are of hand written words.

The list of words for this work aren’t sourced from an existing list, but was one I compiled as a list of nouns relating to abstract but fundamental qualities such as time, light, entropy; states of being and sensations, such as despair, joy, stress. The title refers to the apparently non-physical and subjective nature of experiencing sensation, the way things seem (I feel and experience joy, but it does not have a physical presence, other than what can be inferred from the responses of my body). Through our constant shifting interaction with the world, we form these non-physical abstractions of inner and outer realities.

1c) Faith in Reason

This work evolved from my earliest experiments with the animation of text to form images. It is a meditation on the enlightenment belief in the power of pure logic to act upon the world, the transformation of the abstract thought of symbolic logic to the manifestation of powerful physical objects, tools that enhance our abilities to control and influence the world around us. The glyphs are hand drawn symbols from the Roman and Greek alphabets, and

---

47 See Appendix 2: Algorithms: Brownian Motion Simulation

48 The philosophical term for the subjective experience of sensation is *qualia*. The existence and nature of qualia are seen to be central to the understanding of consciousness.

Describing the Work

include other common mathematical symbols. The objects formed are small prosaic objects: dice, glasses, pliers, needle, key, screw, ruler, safety pin, rubber band, pencil. The objects are all drawn from my immediate personal possessions, and reflect a level of technology that is concrete, personal, and accessible, but also deceptively powerful.

The aesthetic qualities of movement affect perception; in this piece, every position and movement is pre-determined using the classic computer animation technique of cubic splines. The knots along the splines are determined from source imagery using Drawing Algorithm #7; the tonal density of the image is formed through the density of glyphs. The movement is smooth and comforting and harmonious, giving the impression of a logical progression. The movement along the trajectories is not linear, but quadratic, so that the images formed hang suspended for a moment, before moving to the next position.

The title Faith in Reason is a reflection on the enlightenment faith in progress, in the underlying rationality and realism of the ability of mathematics and mechanism to explain and control the world. This classical reasoning - the power of logical deduction - is based in an act of faith, faith in mathematical realism, an objective reality of form that can be apprehended and affected.

1d) Dialectic Seepage

This next work refers to two quotes, one by Noam Chomsky from the 1970s at the height of the acceptance of his theory of universal grammar,49

Language is a process of free creation, its laws and principles are fixed, but the manner in which the principles of generation are used is free and infinitely varied. Even the interpretation and use of words involves a process of free creation.50

49 Chomsky’s theory of Universal Grammar was a dominant linguistic theory from the 1950s until the 1970s. It posits a common set of grammatical structures shared by all languages, with all humans having an innate ability to acquire language. It is currently championed by psychologist Steven Pinker


Describing the Work

The other is by Roy Harris, the progenitor of Integrationism,\(^{51}\)

> *But it is illusory to envisage a language from the start as a self-contained system of signs, equipped in advance to deal with all communication situations that could arise within a given community.*\(^{52}\)

Both quotes are highlighting the fluidity, versatility and limitations of language, but are deeply divided regarding the mental processes of language. Chomsky declares that, although he views language in the mind as having an innate symbol processing system for language (‘its laws and principals are fixed’), the meaning which we attach to words is not fixed, and we are free to use them as we will. This seems to me to be a version of the Cartesian theatre of the mind or the strong A.I. position, where the brain processes syntax, but meaning is ascribed in some other mysterious layer.

Harris, on the other hand, takes a more literal view of the symbolist position, and my interpretation of his position is that language cannot be the result solely of a symbol-processing engine because it would be unable to adapt to new ideas and circumstances. There are flaws in this argument too: it denies the emergent possibilities within systems of formal language. Computer software is a counter-example, based on a fixed system of signs; it is capable of expressing higher-level abstractions not envisaged in the original design of the programming language.

In *Dialectic Seepage*, I take these two opposing views on the fundamentals of language and subject them to a symbol processing system where the words and grammar are reduced to symbols, manipulated and reconstituted, to observe the semantic distortion. This is not quite *Chinese whispers* as there is no communication from one mind to another, and so no process of interpretation, but it is done through a process of artificial intelligence where there is a mechanical syntactical translation of symbols. To do this I constructed a script that uses an existing program called *libtrans*, which is

---

\(^{51}\) Integrational Linguistics emerged in the mid 1980s, it rejects previous linguistic theories such as Chomsky’s, which are based on code or rule based models.

Harris, *Introduction to Integrational Linguistics*.

It shares some similar approaches to Distributed Cognition particularly an emphasis on the creation of meaning through individual context.

Sutton, "Representation, Levels, and Context in Integrational Linguistics and Distributed Cognition".

\(^{52}\) Harris, *Introduction to Integrational Linguistics*. p41.
Describing the Work

used to access existing Internet-based translation services. My script determines which translations between languages are available and then constructs pathways between different languages. For example a pathway might be something like en – de – fr – en – ko – es – en (English to German to French to English to Korean to Spanish to English).

Each of the quotes is taken through a number of pathways of different lengths and then all the words collected and collated to form a collection of around 1000 words. The animation engine is based on the dynamics of damped springs for the foreground text and under-damped springs for the background text; the animation is in three dimensions viewed from an orthographic perspective. The damped spring dynamics for the foreground text (the manipulated quotation) causes it to align and become readable, while the background text continues to move and pulse creating a sea of latent meaning.

Language translation has been mechanised to the extent that the translations are intelligible but idiosyncratic; it reveals something of the underlying syntactical structure of different languages in a way that human-based translation doesn’t. It also allows untranslatable concepts to leak from one language to another, and meanings to drift and reform, like a variation of a literary cut-up technique. Dialectic Seepage is a dialectic engine, taking opposing opinions on the fundamentals of linguistic cognition and subjecting them to semantic torture to reveal a truth.

2) The Transient Taxonomies of Art

The content for this work is based on a web site developed by Robert Belton, called Words of Art. I had been experimenting with force-directed graphs when I saw this

Describing the Work

web site and thought of it as an ontology, a rhizome of cross-referenced terms. I had been pondering the idea of simulating an ecosystem of ideas, a visualisation of a meme tank. I had also been looking at the work of Nodebox Research, and the system they had developed using search engines as a reservoir of consensus knowledge about the world to assign colour and tone to a word.

*The Transient Taxonomies of Art* uses the terms from Belton’s glossary of art terms, and assigns a colour and importance to them based on the Yahoo search engine. The dynamics and interaction of the terms are based on a modified predator-prey flocking algorithm first developed by Craig Reynolds. The work is displayed across a row of three monitors that give a perspectival view into the space of the simulation, in a similar manner to the viewing windows of a large commercial aquarium.

There is a level of irony to this work: I had come to the conclusion that the emergent behaviour of a flocking algorithm (a simple set of rules which generates group dynamics similar to flocks of birds or herds of animals) is similar to other mathematical complexity generators, essentially, a formalism requiring careful tweaking to exhibit the behaviours required. In the title for the work *transient* is a synonym for emergent and *taxonomy* a referent to the hierarchal organisation of the original website and the classification of living forms. Artists who work with


55 Force-directed acyclic graphs are self-organising diagrams constructed from information about members of a set and the relationships between the members.


57 See Chapter 4: Project Development for details
Describing the Work

generative systems would like to create work that is truly open-ended, that can with time increase its complexity, but while emergent algorithms such as flocking are useful and intriguing ways of generating group behaviour, they are not of themselves, creative, and no novel forms will emerge from; rather, the meaning in the work lies in the semantic associations we bring to it, for instance, the chance association of castration swimming with lacanianism that might occur in this work.

3) Communal Sense

This work depicts an ongoing exchange of observations and distillations of knowledge about the world between father and son. The text comes from a collection of quotations about art and science compiled by Jason Newquist.58 The two figures are rendered with the text of the quotations; as a quotation emerges, the individual text elements rise from the surface of the two figures, form the quotation and then move forward in image space. Streams of quotations move forward, and as they become larger, the words are increasingly affected by optical flow measured by a small camera mounted in the screen frame. The viewer must remain still in order to be able to read the quotations, alternately, they can gesture and sweep the words away.

The work is a meditation on the communication of ideas: nurturing an awareness of the world in a child is not a transmission of information, it is a collaborative effort of intention, a dialogue not just of question and answer but of conjecture, anecdote, analogy, leading to a greater illumination for both parties. Knowledge is not a one-way transfer; it is kept alive and grows through a shared examination.

4) Stories about You

We maintain a fiction about our internal cohesiveness, but close observation of inner process reveals not a single I, but a plethora of voices, a weave of narrative constructing and retelling the stories that make up who we are. We change with each moment, we can be many people with different beliefs, capable of things we would never publicly admit. We, as conscious beings, are the sum of the stories that we tell ourselves.

As social beings we have an appetite for the deeds of others, by which we can measure ourselves; there is a prurient satisfaction in reading a truly transgressive confession.

The text in this work comes from a web site called onesentence, which invites people to submit true stories that can be expressed in one sentence. I extracted the stories from the web site and edited them to remove gender and geographic clues, and the more repetitious themes. The work presents a three dimensional space with a neutral surface facing the viewer. The dynamics engine takes these stories and inscribes them on ribbons of text that are released and fall to the floor, where they gradually outline the image of a face. In time, when the face is completed, the ribbons of text are swept away and it begins again.

The image of the face comes from that of a visitor to the gallery whose face has been detected and captured. My intention for the work is that the viewer may be engaged in a public act of participating in the consumption and judgement of others, to be confronted by the shock of seeing an image being formed of themselves. When this occurs, the work provokes a switch from spectator to subject with the intention that the viewer will experience a greater awareness of self, and the multiplicity of others. We all have something to hide; we are all surprisingly alike.

---

Describing the Work

5) Website http://www.billhart.id.au/phd/about

The website contains text notes, summaries and descriptions and documentation of experiments, abandoned works and developmental versions of the work exhibited. No attempt has been made to formalise the writing; it is intended to have the character of a working journal rather than an academic paper or an artwork. The website is not organised in a linear fashion; a minor experimental avenue of this project has been an exploration of tagging and visual representation of personal ontologies (which has not been rigorously pursued). A number of different interfaces for navigating the site have been developed, varying from the now standard tag cloud, to force-directed graphs.

- http://www.billhart.id.au/files/sitemap/navigator.html - uses a force-directed graph built from the tags assigned to each article to build a navigation interface for the site. Because of the number of tags and articles this is not the easiest way to find something on the site, but the interface does provide an organisational view that focuses on relationships between articles, rather than categorisation.
- http://www.billhart.id.au/files/sitemap/vidmap.html - is a refinement of the sitemap navigator, which presents only the video documentation of the project, and with the reduced content, is easier to navigate.
Chapter 3 - Locating the Practise

The history of the digital and computer-aided arts could be told as a history of ignorance against programming and programmers. Computer programs get locked into black boxes, and programmers are frequently considered to be mere factota, coding slaves who execute other artist's concepts.60

This project is broadly located within the field of generative art. Philip Galanter defines the field in the following way:

Generative art refers to any art practice where the artist uses a system, such as a set of natural language rules, a computer program, a machine, or other procedural invention, which is set into motion with some degree of autonomy contributing to or resulting in a completed work of art.61

This is a very broad definition; it could be argued that any artefact that is developed through a process could fit this definition. There exists a plethora of related and sub-genres of generative art which describe art constructed from or with the medium of software: software art, programmer art, digital art, computer art, computational art, processor art, game art, interactive art, new media. The taxonomy of computer-related art practices becomes further confused with the changing use of categories over time as theorists and practitioners have aligned themselves with different factions of the mainstream.

Several broad categories can be made based upon the modes through which the artist engages with software in the development of art works:

1. Artist as software user: where the artist uses a software package to create the art work, for example a digital image made using Adobe Photoshop, or a paper written using Microsoft Word. The computer and associated software is a tool for the production of the work; in many cases the software used (usually proprietary) was developed to simulate a mechanical, chemical or electronic analogue production process (typing, photography, painting, video). The digital process is apparently transparent; there is no inherent digital aesthetic,
but there are signatures that the commercial software and production processes leave upon a work produced with them. This is the common conception of digital art, and began initially with Quantel paintbox systems in the 1980s and gained widespread acceptance with the advent of the affordable personal colour computer. Digital art was allied to many post-modern strategies such as appropriation and bricolage.\textsuperscript{62}

2. \textit{Artist as software critic}: This movement arose during the 1990s in response to the opaqueness of software as the tools for cultural production, and their control by a small number of multinational corporations. This genre, manifested as \textit{net.art} in the 1990s and later more generally as software art, attempted to subvert and expose the underlying layers of software, raising a level of awareness that the apparent transparency is a lie, that along with the power of process and flexibility software provides, there is restriction and control. Sometimes whimsical, often containing strong elements of political or cultural criticism, these artists would hack existing software systems or applications or develop their own processes but, in general, the reference would be to an existing software system.\textsuperscript{63} Artists in this genre include Graham Harwood and Mongrel, Dirk Paesmans and Joan Heemskerk from jodi.org.

3. \textit{Artist as programmer collaborator}: Is where artists collaborate with, or direct, programmers; the artist acting as producer, specifying and communicating an idea to the programmer who would try and interpret it. Commonly based in installation or interactive art, the programmer either creates or customises an


\textsuperscript{63} In his paper from 2002, Cramer identifies two approaches to software art; this aspect he would refer to as Software Culturalism Cramer. "Concepts, Notations, Software, Art". p10.
existing software environment, but is usually not
given creative credit for the work. Artists
working in this mode include Geoffrey Shaw,
Bill Seaman, Char Davies. This was the
dominant mode during the interactivity era from
the late 1980s to the late 1990s, particularly
amongst older artists, and usually required
institutional support. Much of what has been referred to as new media falls
into this category.  

4. **Artist as programmer:** This category is the most complex. It has several
distinct phases; the earliest phase came from technologists producing images
with software in the 1960s, and was called computer art. This was followed
by the artist/programmer in the 1970s. The work was (of necessity)
embedded in algorithmic process, aligned with modernism, and criticised for
its coldness and inhumanity. After a period of unpopularity during the
1980s and 90s, the artist/programmer has been rehabilitated through
software art in recent years. These artist/programmers generally work within
a specialised environment such as the open-source *PureData (PD)* and
*Processing* (which is a significant response to the critical cultural concerns of
artist as software critic) or the commercial *MAX/MSP* or *Adobe Flash*. Artists
working in this mode include Casey Reas and Jared Tarbell. Again there is a
general trend towards formalism but this software art is claimed to be a sub-
genre or successor to the Conceptual art movement of the 1970s rather than

---

64 Florian Cramer has a wonderful diatribe that there is nothing older than new media.

65 Grant Taylor in his thesis gives an insightful overview of the unpopular and largely ignored
computer art movement.

Locating the Practise

to the historically unpopular Computer Art. Aside from the historical revisionism, it is conjectured here that this may be a category error, due to confusion about the use of language in conceptual art and the nature of formalism. However, to dismiss the current manifestation of software art as purely formalist overstates the position and ignores a large body of contemporary work with a significant semantic content. There are numerous examples, for instance, Jonathan Harris and Sep Kamvar’s *We Feel Fine*.68

5. **Artist as systems developer**: The artist develops a program or software environment for the expression of ideas that are not constrained by the given restrictions of a particular software environment, only by hardware and operating system limitations. The artist develops or seeks solutions to explore and express their conceptualisation. Artists such as Harold Cohen and David Rokeby are examples of this category. This project aspires to be in this category, and describes an open environment approach for artists to develop the discipline of expression through software. It is software art that is not about software, but which treats the medium of software as a support for cognition and conceptualisation, allowing it to become like paper and pencil for the process of drawing.

These categories of this taxonomy are not absolute; an individual artist in developing an artwork may incorporate several of these modes of engagement with software. For instance in developing the works for this project, I have used software packages to prepare video and images; collaborated with software developers to fix bugs, extend libraries and port software; and used both closed and extensible programming environments.

---


Locating the Practise

Artists working with software can be split into three broad groups and time periods. The first pioneers were those in the computer art movement that began with the imaging experiments of engineers and computer scientists such as A. Michael Noll and terminating with artist/programmers such as Roman Verostko in the mid 1990s. Computer art was primarily preoccupied with the formal concerns of algorithm and geometry, abstraction and synthesis, and was aligned with late modernism. This corresponds to my category of artist as programmer.

Digital art emerged in 1981 with the development of the first commercial paintbox programs and gained force towards the end of the 1980s with the further development of the colour personal computer and multimedia software. It continues to the present and remains one of the most popular art forms in the age of computers. Digital art saw a shift from programming to software use, with the internal processes of the computer opaque to the artist. This can be described as through the category artist as software user. The formal strategies for image making shifted from the geometry and synthesis of computer art, in which form and surface are explicitly described by algorithm, to the pixels and sampling of digital art in which form and surface are implicit in the data contained in pixels. The technology for sampling and pixel manipulation defines digital art and is closely aligned with many postmodernist strategies.

The third period began in the mid-1990s with the demise of computer art as a critical response to the apparent transparency of software. This was particularly so of the emergence of the Internet, which through creating multimedia and web sites, introduced many artists to concepts of programming. Artist as software critic arose from this critical response to the corporate control of software. Development of programming environments specifically for artists, such as MAX/MSP, Pure Data (PD) and later Processing, led to the recognition of software art as a genre of art practice, and
Locating the Practise

as a consequence, the re-emergence of the *artist as programmer*.\(^6^9\) Computer art was primarily concerned with synthesis and geometry, and digital art with pixels and sampling. Software art combines both geometry and pixels, and the strategies of sampling and synthesis.

With the almost complete digitisation of print, photography, film, video and sound, the approach of *artist as software user* has become the dominant production mode of contemporary image culture. Given this dominance, it is culturally important that *Artist as software critic* continues its critique, but this is not the focus of this project. Significant and interesting work has been made with each of the final three categories (*artist as programmer collaborator, artist as programmer, artist as systems developer*) and the boundaries between these categories are easily blurred. It is self-evident that any artist working with software uses the work of others (a purist would have to return to entering machine code via switches), and that in any medium of expression there is always a recurring interest in formalism. However, it is argued that the maturation of software art lies with the *artist as systems developer*.

Confusing Algorithms with Semantics can lead to Formalism

I’d like to describe a concept I refer to as semantic loading. From an *Integrational Linguistics* perspective, there is no specific distinction between a drawing, symbol or a glyph; it denies in language codes are not autonomous from their context. I define semantic loading as the diversity and strength of associations that a visual object may have.\(^7^0\) The associations the object has may be linguistic or conceptual abstractions, experiential, mnemonic or instinctual. The semantic loading for any visual object is

\(^{6^9}\) *MAX/MSP* was first developed for electronic music in the 1980s, its author Miller Puckette then wrote *PD* in the 90s. Although they were developed for musicians, artists interested in exploring interactivity adopted them. David Rokeby amongst others developed software that extended these environments to incorporate environmental sensors, images, video and 3D models. Artists responded to the visual programming paradigm of these packages, where objects are linked together to create programs. Although I am personally not productive with visual programming metaphors, they do reveal the program as a machine rather than as a text.

*Processing* began development in late 2001 and provides a more traditional textual programming environment.

\(^{7^0}\) This concept is a hybrid of the ideas of mental and physical associations that Damasio puts forward in his *Somatic Marker* hypothesis and the Integrational Linguistics and Distributed Cognition emphasis on individual context.
Locating the Practise

dependent upon the individual context of their current situation and their prior experience. A graphic object transforms from an image to a symbol by a process of reinforcement and resonance, by a collective agreement of the primary use and meaning of that symbol, so that that association becomes stronger than any other, becoming the primary abstraction for that image. The association becomes so strong that we cease to be able to see the image as anything other than a symbol. This shorthand makes the illusion of rapid and reliable communication possible. The human mind is capable of some extraordinary feats. With training, for instance, we can instantly recognise a wide variety of graphic objects as the letter a, and we can deduce the likely meaning of a word from its context. Through practise and habit we have developed such a close approximation to autonomous language codes that, for everyday purposes, we treat them as such.

The semantic loading of a graphic object depends upon the diversity of meanings it can invoke. Two or more words together may either increase or diminish the semantic loading, depending on whether they collapse the range of interpretations or diminish them. In some cases, the juxtaposition of two or more words may cause a convulsive meaning to arise, a collision which blocks or denies one of the primary associations, and forces other unexpected associations to arise. It is interesting that Bill Seaman, although he arrived at it via post-structuralist philosophy, describes a similar conception,

_I write about the idea of fields of meaning and taking the idea of linguistics into a new area where the text has a certain kind of meaning force as a field. Also the image has a kind of meaning force as a field and so does the sound. We also bring a history of our relationships with other environments so we might say that our mind-set is also a field._
Locating the Practise

There is a dynamic summing of these forces we are weighing against each other and the meaning arises out of this.\(^71\)

This conception of semantic loading is different from objective measures of complexity for a system or representation. There are a number of different approaches to quantifying complexity; algorithmic complexity would be measured by the number of instructions required to generate the final object, in which case a fractal would have a much lower algorithmic complexity than the image that represents it.\(^72\) Similarly visual complexity could be measured with reference to entropy, in which case an image of pure noise, which has very high entropy, would have a very low measure of complexity. No quantitative measure of complexity of an image can account for its semantic loading; it would be committing a category error to conflate the two.

Consider the fractal: generating \textit{fractal art} is a populist art activity; a quick search reveals there are 330,000 hits in Google for \textit{fractal art}, but only 260,000 for \textit{impressionist art}. Few contemporary artists take fractal art seriously as an art form; in general, fractals have virtually no semantic loading aside from the iconic representation of the Mandelbrot set and its association with chaos theory, and the \textit{enough} response from the contemporary art commentator. A fractal is a geometric object; it is a geometric object with fractional rather than integral dimension, and is very difficult to visualise without a computer to digitise and map it. Most fractal images have a dimension somewhere between that of a square and a cube. Historically, there was a similar (in


Locating the Practise

some cases quite hermetic) fascination with the concept of the tesseract or hypercube in the early twentieth century.\textsuperscript{73}

Generative art has concerned itself more fully in recent years with the concept of emergence. John Stuart Mill first espoused \textit{emergentism} in 1843 as a middle ground between the extremes of \textit{reductionism} and \textit{vitalism}.\textsuperscript{74} Reductionism takes the view that complex systems can be understood through an understanding of the interaction of their parts: an essentially mechanistic view. Emergentism allows that a complex system can exhibit behaviour that cannot be understood solely by studying the parts – the whole is more than the sum of its parts. The concept of emergence has been revived in recent years with the development of Complexity theory and the demonstration through computational mathematics of emergent properties present in simple systems. A popular example of computational emergence is the \textit{flocking} algorithm developed by Craig Reynolds in 1986 for his \textit{Boids} program, where three simple rules of movement for autonomous agents produce complex behaviours.\textsuperscript{75}

Flocking algorithms are quite widely used by generative artists (my work \textit{The Transient Taxonomies of Art} developed as part of this thesis uses one, as does Casey Reas for his work \textit{Microimage}).\textsuperscript{76} Emergence currently has high value

\textsuperscript{73} Extending the geometric form of a square to a cube, then a tesseract is the extension of a cube to four dimensions.


\textsuperscript{75} Craig W. Reynolds. "Flocks, Herds and Schools: A Distributed Behavioral Model", \textit{Proceedings of the 14th annual conference on Computer graphics and interactive techniques}. ACM. 1987

\textsuperscript{76} Casey Reas, \textit{Microimage}, 2003.
Locating the Practise

amongst generative and artificial life (a-life) artists.\textsuperscript{77} Jon McCormack posits a computational sublime for systems, which, through emergence, surprise the artist by exhibiting novelty and creativity. He goes on to consider the status of computational emergence in comparison to emergence in physical systems,

\begin{quote}
We have also acknowledged that computation, as it exists today, may never be able to give us true emergence, the likes of which we observe in the world around us. As a number of authors have shown, the dialectic of simulation and realization, of life-like and life, are still fundamental issues that, as yet, remain unresolved.\textsuperscript{78}
\end{quote}

Like the fractal, a flocking algorithm has a low algorithmic complexity, but it does have some semantic loading through our instinctive recognition of the swooping patterns it forms as being similar to flocks of birds or schools of fish; it manifests a pattern of behaviour we recognise in nature.

There is no direct relationship between the exhibition of algorithmic complexity and semantic loading or perceived meaning in software art. Algorithms can have distinctive characteristics such as motion, form or texture that become part of the contemporary visual vernacular, drawing upon existing or developing new semantic associations. However, a focus on algorithmic complexity and properties such as emergence does not engage with this semantic loading and implies that the work is solely developed through formal elements such as motion, line, shape, texture and colour.

\section*{Is Software Art synonymous with Conceptual Art?}

\textit{The idea becomes a machine that makes the art.}\textsuperscript{79}

This quote from conceptual artist Sol LeWitt in the 1960s is a seductive description of software art. In recent years it has been argued that the software art movement that began in the late 1990s is a successor or descendent of the conceptual art movement of the 1960s and 70s.


\textsuperscript{78} McCormack and Dorin. "Art, Emergence and the Computational Sublime". p79.

Locating the Practise

In analogy, software art in particular differs from software-based art in general in that it exposes its instructions and codedness. Since formal instructions are a subset of conceptual notations, software art is, formally, a subset of conceptual art.

This quote from Cramer in 2002 makes a clear distinction between software art and software-based art (artist as software critic and artist as programmer) that becomes blurred in later papers. Similarly, the usage of the term conceptual art developed in the 1960s and 70s and focused on a rigorous separation of the idea from its execution - a distinction that became blurred in the 1990s as a result of the association of the term with the Young British Artists and contemporary artists working outside the traditional practices.

This association of conceptual art with software art deserves greater scrutiny. It smacks of historical revisionism, ignoring the fact that computer art was a contemporary of conceptual art and shared many similarities, although computer art did not have the self-reflexivity of the avant-garde. In ignoring the lessons of the historically unpopular computer art, and by making a strong association with conceptual art, there is a danger of misunderstanding and limiting the potential of this medium. Florian Cramer, in another 2002 paper, outlined a dualism between software formalism and software culturalism (again artist as software critic as opposed to artist as programmer), neither of which he claims is a viable genre on its own. Each side has since independently claimed a linkage to conceptual art, both are caught in a self-reflexivity about the medium (which is one of the ways software art differs from the earlier computer art).

---


Locating the Practise

The linkage between software art and conceptual art was explored in the 1990s by Edward Shanken through an examination of Jack Burnham’s 1972 *Software* exhibition, which helped to launch the conceptual art movement (the exhibition displayed works by conceptual artists such as Joseph Kosuth alongside speculative technology projects, from the likes of Nicholas Negroponte and Ted Nelson.83)

Casey Reas, co-developer of the *Processing* programming environment for artists, develops his idea of a software structure and associates it with the instruction art of Sol LeWitt.84 In the *(Software)* Structures exhibition he demonstrates how instructions such as those Sol LeWitt made for his wall drawings of the 1970s can be interpreted through a formal language.85 Reas, with the assistance of several other artist/programmers, interpreted several software structures through different software languages, each programmer/language having a unique interpretation. This was an interesting project and certainly demonstrated how the programmer can give individual expression through the medium of software and the language they use. Nevertheless, essentially it remains a formalist exploration.

I make the case that where software artists confuse algorithmic properties such as emergence with complexity and further confuse algorithmic complexity

---


Locating the Practise

with semantic loading, that this leads to formalism. Whereas computer art focused on algorithmic realism through a reductive formalism, software art focuses on algorithmic realism through algorithmic emergentism.\(^{86}\)

Consider the Composition No 10 by La Monte Young from 1960,

*Draw a straight line and follow it*

Reference to this work has become a common thread in discussions about the relationship between conceptual art and software art. Florian Cramer refers to it as a formal instruction, but is it?\(^{87}\) At first glance it appears to be a recursive algorithm, which could be interpreted through a software approximation, either purely numerically or graphically on a screen.\(^{88}\) But, on closer examination, taking this view is to misinterpret the instruction, to collapse its states of ambiguity and to render it trivial, shedding its semantic load. For example, graphically representing a line continuing to infinity is only one interpretation of the instruction, and how do you follow it? Is it just telling us to draw a straight line of arbitrary finite length? Was it my context as a programmer to read recursion to infinity? How do you follow a line on a computer?

Even to just translate the instruction to a screen as an image, as a literal re-presentation of the work, is to...
Locating the Practise

change its context and semantic loading. In the right presentation and audience context (a gallery and an audience with some awareness of the history of conceptual art) this would be an ironic work; a more faithful representation of the conceptual instruction.

The argument that, conceptual art can be translated to a formal language and software uses formal language, hence software art is conceptual, is to miss the intent of conceptual art, and the expressive potential of software. At the core of the conceptual art of the 1960s was a separation of the idea or concept from the medium of expression, whereas software of any sophistication is deeply enmeshed with the support software and hardware with which it executes. Software is a medium of expression, not a container of concepts. The problem for artists is that engineers, for engineering purposes, have generally designed computer-programming languages for goal such as reliability, consistency or performance, not so often for the expressive qualities of compactness, clarity and power.

It is inevitable that as a new medium of expression develops, it will go through a period of self-reflexivity and obsession with learning the tools of expression. However, an obsession with algorithmic realism and its confusion with semantic loading lead to the aesthetic realms of fractal art. Hence I describe a final category, _artist as systems developer_, with the recognition that this flexible, mutable, intangible medium is like any other. It is through the connections to the physical embodied world that the artwork makes, that it has the capacity to carry a semantic load greater than its algorithmic complexity. An image painted on canvas with oil paints can be as semantically emergent as software. A word processor is the medium for constructing a book which, as an electronic text, has a meaning totally divorced from the complexity of the word processor or resulting document. Code looks like language, but it’s not. You don’t say things with software; you make things that may or may not say something. The previous quotation from Sol LeWitt often has the preceding sentence omitted,

---

89 This is taking a functional approach to programming; what the source code says is irrelevant, it is what the executable does. However, like abstract expressionist painting, you can write code for its own sake. For instance, Perl code poetry was a preoccupation of software art 5–10 years ago. See,
Locating the Practise

When an artist uses a conceptual form of art, it means that all of the planning and decisions are made beforehand and the execution is a perfunctory affair.\textsuperscript{90}

You don’t just write software, you craft it through an iterative, interactive process, which clarifies the idea as the code is constructed. The computer is a ruthless pedant; it tolerates no ambiguity of expression and interprets the smallest detail literally. In regard to software art, I would paraphrase LeWitt as,

Iteratively, the idea guides the crafting of a code object that the execution of in turn refines the idea. At some point this cycle stops and execution of the code is presented as art.

The difficulty with code is that there is no measure or theory of complexity that encompasses semantic association, or aesthetic or artistic success. There is no way to easily ascertain if a software system will continue towards greater complexity and lower entropy with time, and if it will exhibit the quality of \textit{open-endedness}. The computer, coupled with human interaction, whether through the Internet and cyberspace or software art, has this quality. It is an indeterminate system capable of producing novelty but, as yet, the only systems we know which display this open-ended quality we classify as being alive. As with traditional mediums of representation it is the associations the work makes, its content and context, that determine its success, not its algorithmic complexity.

Software Art and Semantics

I would argue that writing software is a process of developing simulation, not conceptualisation. In an analogous way that the name of an object comes to stand for the original object, or a reproduced image stands for the original artwork, similarly simulation can be viewed as an act of deception. We start by concluding that we are all aware of the deception, but as the simulation becomes fine grained, we tend to forget and let the simulation replace its analogue (for example consider the way that photography and video have become digital through software which simulates).


Locating the Practise

Conceptualisation is an act of imagination that happens in the mind. Cognition may be distributed to include the computer as an aid to conceptualisation. Writing software is a process of modelling, of analogising to a mental or perceptual experience. It is an interactive and iterative process; the computer provides feedback through which the analogical model can be tested and refined. But the resulting program is not the concept; it is a cognitive artefact or aid, an expression or simulation of the conceptual image.

So, unless the artist is an essentialist and believes that the algorithm can unfold a level of mathematical reality, then algorithms are a material, the medium or support from which the work is crafted. It is the connections that the work makes to the world that give it its substance.

A Digital Aesthetic

The notion of a digital aesthetic is something that is very difficult to pin down. Computational media has a chameleon-like quality; if a process can be precisely defined, it can be simulated. The visual appearance and engagement by artists with digital media is at any historical instant dictated by the kinds of input/output technology available and the social conception of what a computer is, and is for. For instance the dominant image production tool for the Computer Art movement was the pen-plotter, which ceased production in the mid 1990s about the time of computer art’s demise. Pen plotters are machines for drawing geometry, whereas the dominant image production tool for digital art is the inkjet printer that interprets tonal simulation through patterning dots of ink.

---


The article takes mathematicians to task for their belief in the ontology of numbers. Here I found I had to part company with Harris. Sure there is nothing that distinguishes the signs and symbols for mathematics from any other, but my intuition is that the concept of number is on a different level from other human cultural constructs, and I don’t think you could be a mathematician if you thought otherwise. Thinking on it further, I realised that we all have our articles of faith; Einstein for instance had to be a mathematical constructivist in order to develop his theories.
Locating the Practise

Consider the relationship between digital imaging and photography. The development of digital imaging technology began in the mid 1950s, with the development of graphic displays for SAGE, and the first experiments in scanning images and developing symbolic representations in computer memory. A fascinating aspect to the accounts of this early research is the apparent naivety of the researchers about visual perception, for instance, the initial assumption that tone would be irrelevant to computer image processing.

*The emphasis on binary representations of neural functions led us to believe that binary representations of images would be suitable for computer input.*

Artificial Intelligence researchers in the 1960s assumed that, once an image was digitised, it would be fairly straightforward to identify objects in the image.

Early applications of digital imaging technology were based in scientific/military/government projects, often as a way to analyse photographic data. The development of digital image-processing continued to focus on the concept of an image as a rectangular array of tones separated into bands of primary colour, which is a direct analogue of image capture on photographic film.

With the advent of Adobe Photoshop 1.0 for the Macintosh in 1990, digital imaging emerged as an accessible interactive mainstream activity. A decade and a half later with the development of high quality digital cameras, and digital output direct to photographic paper, the assimilation of chemical photographic processes by digital

---

92 Semi Automatic Ground Environment, based upon one of the first successful computer designs called Whirlwind. SAGE was a USA air defence system, which became operational in 1958.


93 That is black or white.


94 Photographic film is an irregular (but dense) field of light sensitive chemical compounds, filtered into bands of colour.
Locating the Practise

simulation is largely complete, as occurred with audio in the early 1980s and with video in the late 1990s. Even if highly manipulated, so long as the conventions of photography are followed, such images are readily accepted as authentic.

However, discussions in the fine arts about the digital image outside the photographic context often seem to elicit comments expressing concerns about digitally produced images. They fall into two general categories:

- It’s too easy. Ready access to cheap colour printers and software like Adobe Photoshop has democratised image production for the amateur in a way that mass-produced paint sets did in the 19th century. The perception of push button image making denigrates and threatens the substantial body of craft knowledge built around traditional media.

- Uniformity: all digital prints look somehow the same. Colloquially, practitioners refer to an image being a 'shop job, indicating that it is either a seamless montage, or more likely, a heavy-handed melange of obvious filter effects.95

There are two aspects that contribute to this uniformity. The first is that digital print technology (excluding direct photographic printing) is built upon commercial colour reproduction technology initially developed in the 1890s. Images are printed in a single layer using screens of four to eight primary colours. Much research effort has been expended to develop screening and ink technologies that reproduce tone to simulate photographic imagery. The second is that the majority of digital image-making is done utilising Adobe Photoshop. Photoshop is a fluid and rich piece of software but, nevertheless, it is built upon a set of algorithms, and is structured in such a way that (beyond just filter effects) must influence and mark images. For instance, there is an assumed equivalence between the manifestations of an image on-screen, and the manifestations as a digital print when, in reality, the structure and form of the two image manifestations are quite different. The screen has almost continuous tone and the image is perceived by transmissive light, while the print is binary and the image perceived through reflective light. As an analogy it would be

95 This is anecdotal, based on discussions with artists and curators during the 1990s.
Locating the Practise

interesting to see the results of a study on how the grammar checker in Microsoft Word has affected writing style, and whether it has led to more uniform or long term stabilisation of grammatical usage.

In summary, there is no characteristic digital aesthetic; it is dependent upon the technological applications and imperatives of the time. The pen plotter is a device for drawing lines and displaying geometry, suited to the scientific and engineering applications that were the dominant graphical users of computer technology of the 1960s and 70s. The shift to media production applications in the 1980s and 90s required the development of technology for the production of imagery with a photographic quality of continuous tone. While digital ink jet printing has become progressively more sophisticated over the past ten years, it is unlikely to develop into a technology which reproduces images through a multi layered painting process, because there is no social imperative or driving application. However artists always find ways to subvert existing technology for novel applications.

The output devices for this project are flat panel screens, and specialised graphic processors (GPUs). The imperative for the development of both these technologies in recent years has been through entertainment applications, computer games for the GPU and television for the flat panel display. The shape, colour and contrast characteristics of these screens are determined by the shape of the cinematic screen, the utility of displaying two documents side by side, and the rapid action of film and computer games.

A motivation at the start of this project in 2002 was to explore the aesthetic possibilities that the GPU and flat screen offered. This project has made a contribution to this exploration, but development of these technologies continues to be rapid.

Ironically the initial application for GPU type devices was for scientific and military applications, particularly through expensive and specialised workstations from Silicon Graphics in the 1980s and 1990s. There is now a trend for scientists to use low cost domestic gaming equipment as low cost components for building supercomputers.

Locating the Practise

**The Practice of some Artists working with Software**

In this section I survey a number of artists with a long-term practice based around writing software. The intention is to contextualise this project in light of the approach that they have to expression through programming, rather than necessarily on the artworks they produce.

**Roman Verostko and the Algorists**

Roman Verostko was one of the last of the Computer Artists whose work was produced using modified pen plotters.

One aspect of technology is that, as new potentials emerge, there is a period in which a range of diverse approaches are possible, followed by a period of consolidation in which one technological mode becomes dominant. Roman Verostko is representative of a generation of pioneer digital image-makers emerging from high and late modernist preoccupations with form and abstraction.

Roman Verostko was born in 1929 in Western Pennsylvania (USA). He is a historian and Professor emeritus at Minneapolis College of Art and Design (MCAD). After completing a Diploma at the Art Institute of Pittsburgh he became a Benedictine Monk between 1952-68. Verostko had a substantial career as a painter/printmaker for over 30 years before approaching the computer as a medium. His work in his pre-algorist period utilised processes of automatism, or as he states it 'uncontrol and control', 'a kind of dialectic between order and chaos'.

---

Locating the Practise

Verostko began his exploration of generative art in 1980 and he developed and championed the term *algorist* (from algorithm) to describe his and (at the time) most other artists’ methods of working with computers.

*I can now say that you learn a lot about how to draw by writing drawing code and you learn a lot about how to write drawing code by drawing. One of the fundamental problems for the artist who wishes to create pure algorithmic art is to establish a link between the algorithm and his own art making process. When you control the charcoal your own seeing or vision is there. And by vision I do not mean an image; I mean something more – a complex transformative idea about what you experience and see as possible in the finished work. When you hold the algorithm in your hand – when you are in charge as you are with the charcoal – then you also control your vision.*

At the time that Verostko started his exploration there were no commercial graphic applications, only graphic primitive libraries. His engagement with the image-making algorithms was far more intimate than is possible today. For users of contemporary software and equipment many more layers of software (of which they are largely unaware) exist between them and the hardware of the machine; it requires effort and technical expertise to find and access these functions. It’s much easier to build upon the algorithms of others.

The primary output device that Verostko used was the pen plotter. Unlike contemporary raster print devices, which simulate continuous tone pixels on paper, these were vector devices that moved different coloured pens over the surface of paper. For an artist they have some advantages over modern inkjet technology; almost any type of paper could be used, and a wide variety of pens and colours could be fitted to the device. In 1989 Verostko successfully adapted his device to use brushes. These (often temperamental) devices are no longer being manufactured, and their usage is in rapid decline.

---


99 Primitives are shapes such as lines, arcs, primarily developed for graph and diagram plotting.
Locating the Practise

While I don’t share Verostko’s modernist preoccupation with the algorithm for its own sake, I do find resonance in the idea of holding the algorithm in the hand, which I interpret as a need to develop a bodily intuition of the material properties of the software algorithms. The intuition is not in the hand, but in the mind, a sense of the space of possibilities that the software system represents, a process exemplified by Casey Reas:

*I often spend a few days creating a core piece of technical code and then months working with it intuitively, modifying it without considering the core algorithms. I use the same code base to create myriad variations as I operate on the fundamental code structure as if it were a drawing – erasing, redrawing, reshaping lines, molding the surface through instinctual actions. 100*

David Rokeby

*The Giver of Names* is awash in a sea of a language it can manipulate but cannot understand. 101

David Rokeby is an artist based in Toronto, Canada, working with interactivity, software and installation since 1982. He is best known for the development of his VNS (Very Nervous System) for constructing live spaces, where body movements in a space can be used to create interactive environments. 102 He is almost unique among artists in that he has developed this system, exhibited it widely, but also sells it to other artists. His system works with software originally developed by Miller Puckette in the 1980s for electronic music called MAX, but later taken up as a tool by new media artists in the 1990s for performance, veejays, interactivity and robotics. 103 His practice is characterised by a deep technical understanding of creativity with software and electronics, the clarity of his philosophical insight into the medium, and his ability to manifest his poetic vision as an artist through engaging with technology. 104

100 Reas, *Software Structures*  
103 Puckette. MAX/MSP. Computer software.  
Locating the Practise

There are two of his works I respond to particularly that relate to this project. I was immediately struck by the title of *The Giver of Names* (1990 - present); it is a title rich in ambiguity. On one hand it could be a poetic reference to the emergence of consciousness, the ability to name objects around us is one of the key stages of development. Alternatively, it could be an ironic reference to the failures of artificial intelligence, the semantic gap between symbolic and semantic recognition. I suspect it is a wistful reference to his desire to make a conscious poetic machine, and the acknowledgement that its failure is almost certain. It is the only one of his works that does not have a completion date, but is ongoing; it has been exhibited in various versions from 1997 to 2007, in venues in Canada, Finland and the United Kingdom. To experience *The Giver of Names*, the viewer selects objects from a collection in the gallery space and places them onto a pedestal where an image is captured by a camera. The digital image is then processed and analysed to extract shapes, colours and textures. This information is used to generate a word cloud of associations from a database of language and ideas. From these words a linguistic engine constructs grammatically correct, but semantically empty, poetry. The source of engagement for the viewer is that the resulting poetry is not completely meaningless and random; it has reference to the objects placed before the camera, the sentences are syntactically correct, but the meanings are ambiguous and cryptic.

Rokeby describes the background to the development of the *The Giver of Names*:

*In his diaries, Austrian writer Peter Handke at one points talks about “formulation” as the beginning of forgetting. This aligned very nicely with my experience of the tradeoffs that occur when language is applied to phenomena. What is gained is the ability to externalize the experience as a token that can be stored (writing), manipulated (reasoning) and shared (communication). "Coining" a term is an act of power. Adam in the bible is "the giver of names", charged with the responsibility to*
Locating the Practise

bring nature under his dominion through the act of naming. But at the same time, something precious and harder to define is lost. What started as a live, multi-dimensional, organic, and complex interrelation is crystallized into a symbol disjoint from context and experience. It loses its conceptual suppleness. It becomes a ‘stereotype’ of the thing that it is intended to represent.¹⁰⁵

It describes a position very close to my own. Through the process of naming we gain the ability to manipulate concepts on an abstract level. But, at the same time, the other possibilities of categorising and describing the thing we have named are lost. Unless reconnected to the original source of its meaning, with repeated use, a name can become an empty symbol. The fate of millions in the abstract has less meaning for us than the fate of one we know well.

Rokeby also uses the engine developed for The Giver of Names for another installation n-Cha(n)t (2001). n-Cha(n)t won the Golden Nica for Interactive Art at the 2002 Ars Electronica. In describing the motivation for creating n-Cha(n)t Rokeby says,

"The Giver of Names" is awash in a sea of a language it can manipulate but cannot understand. Its plight and its ‘loneliness’ seemed to demand a social group. So I imagined a group of intelligent agents, hanging out in some corner of the internet during their idle time, jamming with their synthetic wits... trying out language on each other.¹⁰⁶

The installation of n-Cha(n)t consists of a room of networked computers creating sound using speech synthesis. The computers also use speech recognition to listen to what is being said, and attempt to respond and create free associations from what they hear. If left alone, the computer network synchronises and the computers chant in unison.

¹⁰⁵ Rokeby, "The Computer as Prosthetic Organ of Philosophy".
Locating the Practise

Again, I remember having a frisson of recognition in reading about this work. I had had a vague idea in the ‘90s for creating prosthetic heads that used speech recognition and synthesis built around a crude linguistic generator to talk to each other. I had speculated that the heads would degenerate into bickering with each other like an old married couple, repeating the same phrases again and again. I wonder if Rokeby was aware when he conceived this project, whether the chanting would be an emergent behaviour….

Rokeby describes the computer as a ‘philosophical prosthesis’, a tool for self-understanding by clarifying the limits of our perception and thinking.¹⁰⁷

Harold Cohen

Harold Cohen (b 1928, London, England), studied painting at the Slade School of Art 1948-51, and was, by most standards, a successful English painter by the mid-1960s (representing England at the Venice Biennale, collected by the Tate Gallery, teaching at the Slade School of Art). A driving force in Cohen’s painting in the 1960s was a struggle to reconcile meaning and representation.

Cohen had many long discussions with Clement Greenberg about meaning and modernism. They agreed upon many things, but on one point Cohen could not agree. He could not accept an art disconnected from meaning and the real world. I interpret this to mean that art was not just a statement about itself as the modernists asserted; art rose out of our experiences in the world. To deny that, was to deny our humanness.¹⁰⁸

An interpretation of this is that Cohen believed that the human mind possessed a kind of symbolic language or notation, that we did not store an image whole in our minds, but as representational tokens (such as how a few smudged lines can suggest a

¹⁰⁷ Rokeby, “The Computer as Prosthetic Organ of Philosophy”.
Locating the Practise

face or figure). Meaning was inextricably tied to the representational strategy used to construct the artwork.

I was always fascinated by the fact that we can look at a drawing and “see” a face, when we are actually looking at a few dirty marks on a piece of paper. That mysterious ability is obviously central to representation, and I’d been stewing over it, not really understanding how it worked, for at least ten years….  

In 1968, bored and frustrated with the British art world, Cohen left for a year long residency at the University of California at San Diego, where he met computer scientist Geoff Raskin who introduced him to computer programming at the age of 40. Cohen remained in San Diego becoming a professor at the University of California. In 1971 he exhibited A Computer-Controlled Drawing Machine. From 1971-1973 he was a guest scholar at the Artificial Intelligence Laboratory at Stanford University. It was there that he conceived and developed the first version of his computer program AARON, which he continued to develop for the next thirty years.

When I met my first computer in 1968 I had an intuition that I might be able to learn more about this mystery by programming than I had learned in twenty years as a painter. I couldn’t persuade the computer to produce images -- marks to which the viewer would assign meaning -- unless I could propose a mechanism capable of generating the images.

AARON has had several stages of development, starting from being able to only draw lines, through to colouring and shading by the mid 1990s. AARON is based upon Cohen’s observations of how children draw,

At that point I did, indeed, look very closely at the way young children draw. It seemed to me, from having watched my own children when they were very small, that


110 Cohen. "Aaron the Artist -- Harold Cohen"
there's a point at which they will produce a sort of closed, containing shape for their scribbles, and that this point corresponds to their notion that they are making a representation -- that the marks they make "stand for" something. AARON's earliest efforts involved being able to draw an enclosing form around a scribble.\textsuperscript{111}

AARON is based upon two types of knowledge; one is the declarative information about the structure of real world objects, tables, pot plants and people; the other is a set of rules and procedures for building a representation. The current version of AARON contains around a thousand such rules.

Understanding of human cognition and the nature of intelligence have changed dramatically in the thirty years that Cohen has been developing AARON. In the early 1970s when AARON was first being developed A.I. research was very active, receiving substantial DARPA (military) funding. Computers had developed from being numerical to symbol processing systems and this was the paradigm that most A.I. research adopted, the reduction of knowledge to precise rule-based symbolic or linguistic systems. AARON has often been perceived as exhibiting intelligence and creativity; one of the most interesting aspects of the AARON project is the changing way in which Cohen and other commentators refer to the program. In the '70s and '80s the program is often credited with having intention.

The significance of AARON is not that it makes art, or that it demonstrates some kind of machine intelligence.\textsuperscript{112} In more recent times Cohen himself

\textsuperscript{111} Cohen. "Aaron the Artist -- Harold Cohen"

\textsuperscript{112} It is worth noting that the form of image making that AARON has produced has changed dramatically over the three decades. Initially AARON only drew outlines, to which Cohen
Locating the Practise

acknowledges that AARON’s output is original (in the sense that each image is different), but not creative (in the sense that AARON could add unexpected elements to an image, or develop a novel mode of representation). AARON is not an autonomous agent. Cohen adopts the strategy of many generative artists, which is to use the artefact as a tool to generate a large number of possible images from which he selects a small number for further development and exhibition. AARON is meta-art, an artefact which produces images; its significance comes from Cohen’s lifelong investigation into the way that humans make images and like many A.I. artefacts, it defines us by what it can and can’t do.

John von Neumann in 1948, said that it was impossible for a machine to think:

You insist that there is something a machine cannot do. If you will tell me precisely what it is that a machine cannot do, then I can always make a machine which will do just that! 

This illustrates concisely what has been learnt through Artificial Intelligence research; general intelligence consists of those things that you can’t define precisely.

**Summary**

I have located this project within the broad discipline of Generative Art, and the more recent category of software art. In the context of this project I define software art as an art practice, in which writing software is the medium of expression.

The key insights of Integrational Linguistics and Distributed Cognition – the context dependence of communication, the non-coded nature of thought, and the distribution of cognition to include artefacts – provides a new perspective for examining software, programming and expression. During the 20th century we made a machine that we thought was in our self-image, but have since discovered is not; that insight can be

---

then added colour; this early AARON was very much a generative system. It wasn’t until the mid-1990s that Cohen developed a system for AARON to generate colour.


[114] Von Neumann is considered by many Americans to be the architect of the first computer. He also developed the concept of MAD (Mutually Assured Destruction).

Locating the Practise

used to probe and question our assumptions about language, code, algorithm and thought. We should be cautious about the relationships we make between code and language, algorithm and meaning, concept and program.

It has been argued that the materials of software art, algorithms, are only meaningful through the associations they make with the physical world; that there is not a direct connection between algorithmic complexity and semantic loading, and to solely pursue algorithmic realism leads to formalism.

The relationship between Conceptual Art and software art is complex; the linkage that has been made between the scores and instructions of Conceptual Art and the programming instructions of Software art has been questioned. The essence of the former is the ambiguity of natural language, whilst the latter is formal and precise. As carriers of meaning, they operate on different levels of description; natural language is inherently meaningful through the web of associations each word has. By itself, code is meaningless; it is the completed code object that has the semantic capacity for association. Software art needs to be considered in a broader range of contexts; it is a mistake to ignore its relationship to the stigmatised Computer Art.

The problem of aesthetic uniformity in technological art works has been raised. Both the contemporary software imaging and hardware display technologies available determine the appearance of the work. This project relies upon LCD display panels, 3D accelerated graphic cards and the OpenGL graphics library, which determine the physical size and shape of the works, but also the underlying appearance based on geometry transformations, lighting and shading algorithms common to this technology.

The practices of three artists, Roman Verostko, David Rokeby and Harold Cohen have been examined in the light of these ideas. These artists represent three distinct approaches to writing software to create artworks: Verostko, representative of the Computer Art movement with his strong essentialist focus on the algorithm; Cohen, with his thirty year project to construct a drawing machine so that he can understand the cognitive processes of drawing; and Rokeby, who writes software for others to use, but recognises and explores what the computer can tell us about ourselves through our failings. All have been influential in the development of this project.

Software art is a collective project that explores writing software as a means of creative expression; there are potentially significant philosophical insights that can
Locating the Practise

come from this project. In this chapter I’ve begun an exploration of some of these, through looking at the discourse around the discipline and through an examination of the practices of some other artists.
Chapter 4 - Project Development

This chapter describes the temporal development of the project and the works that comprise the thesis. The descriptions of the work in Chapter 2 are a summary of the completed and contextualised work, whereas the descriptions here are of the developing work and the contexts at the time. As much as possible, the technical detail of analysis and method are contained in the following chapter. To view documentation of the work being discussed, this chapter is most conveniently read in conjunction with the graphic time line on the web site.115

Prologue to the Project – Static to Animated Images 1996 - 2002

My interest in the interaction of text and images has been long term. From cartoons to my earliest video art, I juxtaposed images with text and spoken word to amplify or focus the semantic content. In the mid 1990s my practice turned towards constructing large two-dimensional digital images. Ironically, since digital-imaging software makes it easy to integrate text and image, I found text to be an uncomfortable element.

My early explorations of the computer as a medium for image making led me to question how the opacity of the software and associated print technology influenced the images I could make. This was at a period where there was simultaneously, an enthusiasm for digital art and scepticism about the quality and longevity of digital prints. It has been implicit in the engineering development of digital-imaging technology that the goal is to simulate photographic or filmic representation. The artefact of the digital image is often confused with the process. For instance, at one stage in the 1990s a photograph was an image printed onto light-sensitive paper, which distinguished it from a digital image printed via an inkjet process.

The digital image is a reductive conceptual abstraction: grids of eight bit numbers corresponding to the colour channels and layers that describe the image.116 There is a

Note this interface requires that a PDF viewer be installed.
116 Integers with a value between 0 and 255
Project Development

vast range of possibilities for how images can be modelled digitally, and processes through which digital images could be turned into physical artefacts. Early in the development of digital imaging technology the objective became to construct a transparent simulation of the analogue photograph. As Magritte's pipe reminds us, we tend to forget that a photograph of a painting is not the same as a painting, and that a digital-imaging model that simulates the complexities of layered paint, or a printing technology that works with layers of pigment and medium, are also possibilities.

At the time it seemed that, for all its potential, digital imaging also represented the potential homogenisation of image culture. All images would be created and produced using the same tools and technology; in other words, for an end-user the technology seemed sealed and hermetic. It was within this context that I started to think about ways to intervene in the digital-imaging process.

The most accessible point of intervention was in the digital-imaging system that translated the pixels of the image to patterns of ink when printed. This led me to consider that if the resolving power of the digital-image grid was high enough then the grid-like structure of the image effectively disappeared, and other mesoscale structures could be imposed, creating the equivalent of the mark in traditional media. So, rather than focusing on detail to hide the signature of the digital process, I experimented with taking low-resolution images and adopting alternative rendering methods. These affected the apparent surface quality of the image and added the potential for multiple readings to be taken from the image. Initially, this experimentation was done through Photoshop tricks, but eventually it led me to writing scripts to control the imaging tools. This was easier to do using open-source software - The Gimp imaging tool and the Perl scripting language running on Linux computers - than with commercial tools such as Adobe Photoshop and Mac OS 9, where the stuff of software is only accessible through the Graphic User Interface (GUI).

I found that this type of experimentation sustained my interest and imagination in the image-making process. There was an excitement about having to intuit how a process that took several hours to construct an image would translate from screen to print.

117 This work was supported by a large ARC grant entitled Characterisation of digital mesostructures in computer generated images (1998-2000) with Professor Geoff Parr, and Dr Mary Scott.
Project Development

The final images were huge for the time (several hundred megabytes) and impossible to see on screen because of their structure and size. The relationship between the print and the image for these works was the reverse of the normal: the screen lacked the resolving power to represent the image; instead, the printed image was the artefact, not a facsimile of the computer-generated work.

Eventually I came to use text as the mark for the image, gradually refining and adding observations about how images are drawn. The culmination of this research was an image of a rumpled shirt called *I think therefore I am not sure…* in which the text used was from Descartes’ *Meditations on First Philosophy*. The technique used to draw the image was straightforward (See Appendix 2, Drawing Algorithm #1)

I had for some time wanted to return to working with the moving image, and thought about how I could use a similar system to draw a sequence of images that could then be put together to create an animation. I knew that this would be quite hard to do, because

---

Project Development

rather than having a large canvas of up to 30,000 pixels across, a screen (at the time) could only have 1024 pixels across. There would have to be far fewer words and, consequently, the imagery would be harder to distinguish. I selected a range of formally simple source imagery, and a random selection of words from a dictionary. Initially I just wanted to see what happened (see Appendix 2, Drawing Algorithm #2).

The animation took much longer than I expected to render: after six weeks and having to check on it several times a day, it had only rendered half of the animation, so I stopped the process. The results were intriguing, but it was a long way from being a finished work. I concluded that I would have to find a way to make the image-rendering process considerably faster.

The final prelude to this project was a collaboration with Professor Patrick Clancy, Professor and Chair of Photography and Digital Filmmaking at Kansas City Art Institute, on a project of his called The Writing Machine. It consisted of a computer interface that could be read. The text was constructed from a stack of stories that interacted with each other using data from weather sensors. I spent quite some time figuring out how to visualise the interaction of the texts which, as a last resort, led me to brushing off my programming skills and writing a program in C which would use the OpenGL graphics libraries to run the display. By undertaking this project, I developed the skills to be able to see how I could animate thousands of text elements in real-time.

General Methodology

Casey Reas (co-author of the Processing environment) has developed the notion of sketching for describing the process of developing software-based art works. He relates

---

119 http://toolshed.arthbt.utas.edu.au/billhart/phd/mv/words
Project Development

the notion of sketching not only to the preparatory phase of a work’s development, but also as an iterative process, where the work is progressively defined and refined. This occurs as feedback is generated between the writing of the software and its visual manifestation, which then leads to the further modification of the software. This is a very different approach to the traditional top-down view of software engineering where the problem is identified and then progressively defined until it is completely specified.

It is necessary to sketch in a medium related to the final medium so the sketch can approximate the finished product. Painters may construct elaborate drawings and sketches before executing the final work…. To sketch electronic media, it’s important to work with electronic materials. Just as each programming language is a distinct material, some are better for sketching than others. ¹²⁰

The sketching methodology that Reas articulates is relatively recent, but is similar to the methodology developed through this project.¹²¹ The discipline required for sketching (as for all visual representation) is to have an intent that guides the evaluation of each stage of the development process. This may be a conceptual image that the artist wishes to manifest (how do I make something that looks and moves like this?), or in the form of experimental exploration (what happens if I take this and this and put them together?).¹²² Often the parameters of this intent are difficult to define in words, but are easily recognisable through intuition.

Sketching is one metaphor, but there are also aspects of the developmental process that could be described as sculptural. Once a form has emerged through a sketching process, there may be a stage of quite deliberate craft that is less iterative, of selecting the appropriate software materials and building the armatures on which the work will depend. A highly iterative process of tweaking and tuning follows. This involves exploring the space of possible behaviours of the work, developing an intuitive sense of


¹²¹ The first version of Processing was released in late 2002. Reas began articulating his software sketching metaphor in 2003. This project developed in parallel with Processing, and confirms and expands some of the insights about expression through software that have come from that project.

¹²² I use the term conceptual image quite often in this chapter. By conceptual image, I am referring to a non-verbal mental image / thought / sensation / idea that an artist may have for a work.
Project Development

that space in order to select the values of constants and kludges that express the intention of the work.\textsuperscript{123}

The works here are all developed through several distinct stages with iterations between one or more stages as the work is refined. Each stage may involve selections of tools and methods, and a process of further refinement, not necessarily always in the order described here:

- \textit{Conceptualisation} - describing, and clarifying the concept (may involve background reading, drawing diagrams)
- \textit{Survey the Field} - surveying what has been done that is similar, possible starting points and strategies, toolkits, tutorials and examples. All software is built on the intellectual work of others, although there are often conditions and ethical considerations in using the work of others. In general, I choose to use open-source software and often start the development process with code from an example or tutorial.\textsuperscript{124}
- \textit{Content Selection} - selection of source media and imagery (may involve searching, reading, collecting imagery, refining imagery)
- \textit{Data Preparation} - materials are processed and manipulated into a form suitable for visualisation. Usually this involves higher order software processing, and is where the semantic content is connected to the formal structure of the work.
- \textit{Visual Manifestation} - the animation engine that manifests the simulation space and generates the dynamics of the work.
- \textit{Reflection} - The work progresses to a stage and then stalls. Reflection occurs at all stages, but inevitably, a point is reached where there is no clear way forward. At intervals the work needs to be assimilated, digested, cogitated upon, its strengths

\textsuperscript{123} In some works, there may be up to thirty arbitrary parameters to be defined, which will determine the speed, stability, the quality of movement, the type of interactions. I have found the concept of phase space developed by physicist William Gibbs in the early 20\textsuperscript{th} century useful to describe this. Phase space is an n-dimensional space constructed from all the variables that describe a system. The behaviour of a system can be described by a form within that n-dimensional space. The artist intuits this form and selects ranges of parameters that give it the characteristics that suit their intention.

\textsuperscript{124} It is difficult to know at what stage a piece of code becomes your own. When none of the original code is left? When it no longer has the same function? Or is it when you have contributed something new to it, that wasn’t implied in what you started with?
and shortcomings acknowledged. Eventually there is a sense there is no more that can be done and that a new insight is required to further develop the work. The discourse around the work becomes precise in conjunction with the work’s resolution. The description of the work in Chapter 2 was made after the work had been largely completed; the discussion here is based upon how the work developed. I hold the view, and this was a premise of this project, that it is through the process of making that an idea is expressed, because the whole thinking self is engaged. When an idea is articulated, the label or terminology we use to describe it becomes a symbol, a substitution for the idea itself, and the connection with the original conceptual image can be lost, along with the ambiguity and nuances that may not have been fully expressed. Perhaps this is an attitude that comes from training in experimental practice in the physical sciences, where knowledge of theoretical and practical principles is necessary to conceive of a worthwhile experiment, but where, in order to avoid bias, results are only analysed against theory once the experiment is completed. Similarly the works are rarely named, except for a short working title, until they are nearly completed.

2003 – Exploring the Possibilities

The first year was spent in defining the topic and the methodological approaches through constructing a variety of exploratory works. A description of the concept, approach and outcomes for each of these works follows.

Memetic Variations – 2003

Conceptualisation

This is the earliest manifestation of a work, which after several stages of subsequent evolution, has been resolved as The Conditions of Ambient Cognition in the gallery submission. My initial research proposal from early 2003 outlined the following concept for this work,

*I have long been interested in visual representation as a form of heuristic knowing, as opposed to forms of knowledge based upon logical construction. The project will utilise memetics, or the theory of memes proposed by Richard Dawkins, to describe cultural evolution by selection of adaptive ideas, as a framework for the investigation.*

A personal motivation for becoming interested in art was a realisation that some art works seemed to be able to express a philosophical position, a synthesis of a range of
Project Development

ideas. For instance, looking at the history and evolution of ideas, key moments can be isolated in which a visual representation can represent the truth or untruth of a historical moment. I think of the unsettling paintings of De Chirico projecting the loss of certainty about the nature of time and space in the early 20th century.

Just prior to conceiving this work I had re-read The Selfish Gene and was intrigued with giving form to an idea that borrowed some of Dawkins' concept of a meme (a unit of cultural information), and Thomas Kuhn's theory of paradigm shifts.\textsuperscript{125,126,127} The conceptual image was of a range of concepts coalescing to form a new complex idea, which although initially fuzzy, would eventually clarify and gain definition. As time passed, the clarity would become stasis, until eventually, it became so rigid that it collapsed under its own inertia. This was something I'd observed happening in my own mind, and it seemed to describe the flux of fashionable ideas and cultural forms, as well as scientific theories. I wanted to find a way to give this insight form, to make it concrete, not just so that it could be communicated to others, but to examine it more closely myself.

In observations of my own internal process I had experienced this phenomena on numerous occasions. An insight would appear (usually when I was tired and relaxed) as a sense image - the strange combination of image, emotion and form. These moments of revelation might occur on several occasions for the one insight, gradually clarifying and defining the thought, and in between, the conscious mind would be busy at work, filling in details, exploring the ramifications, pondering how to approach


\textsuperscript{126} A meme is a unit of cultural information; an idea, a value or a habit. A meme complex is a related group of memes, such as an ideology or religion. Philosopher Nicholas Humphreys anecdotally is said to have suggested the idea to Dawkins. A criticism of the concept is that it perpetuates a deterministic mind / body duality, memes for the mind, genes for the body.

\textsuperscript{127} A paradigm shift is a fundamental change in approach or underlying assumptions. It is generally associated with a change in dominant theories in science, the shift from Newtonian to Relativistic mechanics for instance. Examples in recent history are, Chaos and Complexity theory. I would argue that currently there is a paradigm shift occurring in our understanding of cognition and consciousness.

the problems associated with the idea, translating the sense image into concrete concepts.

As an insight becomes more defined through language, it can also become less interesting. A time can come as it turns over in your mind, where you realise you have lost something of the original insight, that the words or the image that you have constructed to contain the insight have become symbols that substitute for that insight, but the original insight is lost. I had observed that through making, the act of poiesis, that the manifestation of insight into visual form before it becomes stale, could contain something of the original thought. The difficult balance, the art, is to find a vehicle of poetic expression that allows for a tension between ambiguity and clarity: the expression of insight without being so generalised it says nothing; and yet not so defined that it becomes didactic or illustrative. It is the discipline of drawing an object you can only see in your peripheral vision: to look too intently is to destroy with the gaze of attention.

My ambition for this work was to explore and express aspects of the processes of thought, without compromising the original insight through a literal translation. I found that the development of the work became intertwined with the processes of its production, so while the bulk of the technical analysis is in the next chapter, inevitably there is also some here.

Surveying the Field
Project Development

The rendering of images with text bears some resemblance to a calligramme. The calligrammes of Apollinaire, for instance, encapsulated poetry in a pictorial form, where there is a direct relation between the textual content and the word picture that is created to represent it. In recent years there have been a number of animated or interactive calligramme works, for instance the Typographic Illustrations of Evan Roth and Mark Asare or the Type Drawing tool of Hansol Huh. Calligrammes can generally be read, while in Memetic Variations the text is dense and unstructured. The readings that come from the animation are generally from a chance conjunctions of words and, in that sense, the conception of this work is closer to the literary cut-up techniques of Tristan Tzara, or William Burroughs and Brion Gysin.

Data Preparation

An initial range of sample images were selected with a range of visual complexity, including small man-made objects, symbols, words, formulas and faces. The aim was to develop a sense for the type of imagery that would be discernible and how it should be prepared.

128 Set to music and authored in Macromedia Flash these use different typefaces to draw portraits of famous figures.
Evan Roth and Mark Asare, Typographic Illustration, 2003, New York.

129 At the time of writing this was temporarily unavailable, but it was a tool that facilitated the user creating calligrammes.
Hansol Huh, Type Drawing, 2005.
Project Development

The text came from a random selection of 5000 words chosen from the standard unix dictionary file: /usr/dict/words

Data preparation software calculated animation keys, by determining the locations for each word in the image; using a similar algorithmic process to the one I had developed for drawing the image I think therefore I am not sure… (Appendix 2: Drawing Algorithms #3 and #4). The two algorithms were used to process each image and the result stored as key files. A Perl script, using the ImageMagick toolkit, rendered an image for each of the 5000 words used in the animation.

Visual Manifestation:

For this work I started with the tools I had used for The Writing Machine - the C++ programming language and 3D hardware accelerated OpenGL libraries. The OpenGL tutorials of Jeff Molofee were invaluable in developing the engine.

The animation is controlled by a combination of randomly selected animation keys and a dynamics engine. This was the first time in fifteen years that I had brushed off my knowledge of numerical methods of classical dynamics, and the Euler

Using the terminology of animation, a key is a key frame where the attributes (position, rotation, scale, tone) of a glyph are defined, with a type of tweening to occur between keys. The tweening is an algorithmic process that gives the animation its dynamic qualities. For instance, most computer animation uses by default cubic spline based tweening that has a distinctive smooth motion, as opposed to jerkier motion that comes from linear tweening.

Project Development

integration method I used was inaccurate and prone to instability.\textsuperscript{132}

The acceleration of an individual word was stronger the further away it was from its key point. This caused the text to oscillate around the key point, gradually being damped by friction. Another two unconnected equations controlled the spin of the text, and its movement in the Z plane (in and out of the screen).

As OpenGL has no way of automatically rendering transparent objects correctly, it was necessary to sort the depths of the 5000 text points so that those furthest away from viewpoint were drawn first.

Reflection

This program took several months to develop, and I was pleased and surprised to see how well it worked. Animating 5000 separate words at once seemed like quite an achievement in 2003. However, the text tended to spin annoyingly, and the program stability was very sensitive to any modification. I had been struck by a quote from Henri Thoreau that technologist Bill Joy had used in an article warning of the dire consequences of the unfettered development of technology.\textsuperscript{133} The quote was ‘We do not ride upon the railroad, it rides upon us’, and I incorporated a procedure for not only rendering images with text, but for also rendering strings of text (the text would become white, and the other text would become black and recede into the background).\textsuperscript{134}

5000 pieces of linear text are not enough to form recognisable detail in imagery; it is only possible to reproduce outline shapes and some tonal modelling. The documentation is crude, but gives an impression of the work.\textsuperscript{135} This was a successful

\textsuperscript{132} Numerical integration techniques for approximating dynamical systems only work under a limited range of conditions. If time or space quantisation is too large for the forces being simulated then the numerical scheme will become unstable and blow up. The program crashing or everything disappearing from the screen usually indicates that this has happened.

\textsuperscript{133} Bill Joy, "Why the Future Doesn’t Need Us," Wired Magazine 2000.


\textsuperscript{135} The movie is made using a screen capture program as the animation runs, and consequently, has a low frame rate compared to later documentation.

http://toolshed.artschool.utas.edu.au/billhart/phd/mv/memetic-variations-cplusplus
Project Development

experiment and test bed, but was a long way from my initial conception or from resolution in its own right. I identified several issues that needed to be addressed to further progress this work:

• A more sophisticated and reliable dynamics engine.
• Image-rendering algorithms that rendered greater detail.
• The programming environment for C++ was inflexible. It was hard work to add in new capabilities, and easy to lose the sense of what the code was doing.
• The relationship between text and image needed far greater consideration.
Jitter - 2003

Conceptualisation

Rather than try to resolve Memetic Variations immediately, I tried a different approach to drawing with fragments. Rather than creating an animation based on predetermined keys, this would be process-based. The fragments would interact with each other, the interactions being influenced by a field derived from an image. The fragments would have a tendency to repulse each other; the imposed field would mediate this repulsion. Where the field was strongest the fragments would coalesce, so forming an unstable image. I wanted to make a connection between the physical world, and the abstracted nature of our conviction of unity. The metaphor would come from the detritus of the body that we constantly shed and exchange with our environment, contrasted with images of physical agency.\textsuperscript{136} The intent was to explore the relationship between this seething mass of physical stuff that makes us manifest, and our implicit belief in a transcendental self.

\textsuperscript{136} We shed about 100 hairs per day, 7 million flakes of skin per minute and have around ten times the number of foreign microorganisms in our gut as cells in our body.

George W. Ware, Reviews of Environmental Contamination and Toxicology (New York: Springer-Verlag, 2003). p154.
Project Development

Data Preparation
I thought that this work would be straightforward, so I didn’t look deeply into how other people might approach this type of simulation. I continued to use similar tools and methods to Memetic Variations. Data preparation was quite simple; a script to convert an image file into a coarse grid of numbers (54 x 28 elements). I only used one image in the work, a hand. Scanned snippets of pubic hair were used as the image fragments. Pubic hair forms round curls and is not too detailed.

Visual Manifestation
This was written as a C++/OpenGL program, with 2000 animated elements. To avoid an \(N^2\) calculation, the screen space was divided into a grid of 54 x 48 elements, and the fragments were binned.\(^{137}\) That is, a grid is imposed and each agent is assigned to a grid square, so that only the interaction between fragments in adjacent grid squares needs to be calculated. Acceleration from each of the adjacent grid squares is calculated from the difference of field intensity (derived from the image to be rendered) reduced by the proximity of other fragments. These are summed to calculate an accumulated acceleration vector and added to the fragment velocity. Friction is applied so that the total energy in the system doesn’t escalate.

Reflection
At the time I thought that this work failed. Viewing it again after several years I’m not so sure.\(^{138}\) The fragments never convincingly formed an image, however, I only tried one image – that of an open hand. The image would come to the brink of forming a recognisable representation but, in retrospect, there is an enigmatic quality

---

\(^{137}\) A computation problem with agent-based simulations occurs where potentially every agent can influence every other agent. So the distance and interaction between each pair of agents has to be calculated. For instance, when there are 10 agents, there are 90 interactions to calculate, but when there are 100, then there are 9900 interactions. Problems like these are said to be \(N^2\), as the number of calculations increases as the square of \(N\).

Partitioning the space to reduce the number of interactions to calculate, can reduce the number of calculations to \(N \log(N)\), i.e. the number of agents times the natural logarithm of the number of agents. So our 100 agents only require 460 interactions to be calculated using these methods.

\(^{138}\) This movie is again made using a screen capture program; the actual program forms a more defined image than can be seen here.

http://toolshed.arthbt.utas.edu.au/billhart/phd/jitter
Project Development

to this. An artefact of the dynamical system caused the fragments to form into diagonal rows; I later realised that the way I had constructed the system was a form of cellular automata, and the artefact was related to the way the system was iterated. I felt a frustration with the rigidity of the C++ environment, the system of binning and calculation was complex, and it was difficult to try alternatives. It was like being bogged down in detail and losing sight of the objective.

Moebius - 2003

I constructed this work as a way of clarifying my conceptual premises about knowledge, knowing and being. It is an animation constructed in Cinema 4D.\(^\text{139}\) It consists of 3 inter-linked moebius strips with each strip displaying a proposition:

\[
\text{The Technological Imperative: The Conjunction of the disappearing display and accelerating real-time graphic performance will have the biggest influence on visual culture this century.}
\]

\[
\text{Knowledge and Knowing: The ability of visual representation to act as a form of intuitive subversion (knowing and insight) against conventional codified frameworks of knowledge.}
\]

\[
\text{Form, Representation and the 'Digital Aesthetic': software is a set of abstractions, which with display devices create the aesthetic limitations - There are many other possibilities.}\(^\text{140}\)
\]

The moebius strip is a geometric oddity: it is a shape that exists in three dimensions but only has one surface (a sphere for instance has an inner and outer surface); it is formed by taking

\(^{139}\) A 3D modelling and animation package.


\(^{140}\) http://toolshed.arthbt.utas.edu.au/billhart/phd/moebius
a strip and twisting once and then attaching the two ends. The application of a text

texture to the surface creates a looping sentence that covers both sides of the object.

**Nodelands – 2003**

**Conceptualisation**

In the early stages of this project I considered a number

of alternative directions for exploring relationships

between knowledge and knowing. Constructing tools

that facilitate relationships and structure to emerge

from a set of concepts was appealing. I had been aware

of the Visual Thesaurus for some time and had become

interested in *free tagging* as a way of organising

information. I had looked at a number of tools for

constructing force-directed graphs, that is, visual

representations that use a dynamical system to organise

the layout that consists of nodes and their linking relationships. The nodes repel each

other, and the linkages are like springs that pull the nodes together.

**Surveying the Field**

This use of software to explore and make

explicit relationships in information is

generally known by terms such as data

visualisation and data mining. In recent

years it has come to be termed *Information

Aesthetics*, and has moved from the domain

of science and scientific visualisation to the

more design-oriented visual


---

141 Originally *Visual Thesaurus* by Plumb Design released in 1998. This has evolved into a

commercial product.


142 Rather than constructing an initial taxonomy and then using that to categorise information,

*free tagging* allows for the addition of new categories as required. To be useful it still requires
discipline so that multiple similar or non-descriptive tags do not proliferate.
communication. There are numerous examples that span the spectrum from applied to speculative art.

The conspiracy maps by neo-conceptualist Mark Lombardi have a clear formal relationship to force-directed graphs. Even though they are drawings and not software, the principles of organisation and presentation are very similar. Lombardi’s drawings, which date from 1990, appear to predate software for force-directed graphs.

Ben Fry (co-author of Processing), has a practice focused in data visualisation. An example of this is his work Valance which explores structures inside books. A similar but two-dimensional work is Textarc by Bradford Paley as an online applet to interactively analyse character and plot in Hamlet or Alice in Wonderland.

**Visual Manifestation**

One of the earliest implementations I looked at was TouchGraph, a java-based framework, which was clunky and ugly as most java graphics were at the time, looking like something designed by

---

Project Development

engineers for engineers.\footnote{145}{Only older versions of the source code are now publicly available. This site is the commercialised version.}

I started from some source code provided by Jared Tarbell in 2001 on his website. Called Pond Code it is based in Macromedia Flash, that has a high-quality drawing engine.\footnote{146}{TouchGraph LLC, Touchgraph, 2001-2007, Available: http://www.touchgraph.com/} Tarbell’s code was useful as a starting point primarily as I wasn’t very familiar with programming in Actionscript (the programming language for the Flash environment). Nodelands took as input an xml file that contained a node name, a description, an optional image file and a list of related nodes.\footnote{147}{Jared Tarbell. Pondcode. Computer software, 2001. Available: http://www.levitated.net/daily/levPondCode} The xml file could be created with a text editor, and added to incrementally. It became a kind of presentation tool, in which the presenter could wander across a range of topics.

Reflection

A problem with this version is that, on newer machines, it is too quick to jump to the presentation mode, and allows more than one image at a time to be presented so that an irritating jumble of slides can accumulate on screen. There has been a lot of research and experimentation with force-directed graphs in recent years but, as yet, they lack something which makes them a truly fluid and intuitive way of navigating and understanding the relationships inherent in a body of information.

Information-visualisation projects and techniques have been informative to this project, but have not been pursued as a line of research. The focus in this project has remained on exploring aspects of cognition through visualising conceptual images, rather than developing methods for aiding cognition through visualising relationships.

Towards the end of this project, force-directed graphs were revisited as a visualisation front end for the web site. Once again, the implementation is through Flash, this

\footnote{148}{eXtensible Markup Language}
Project Development

time built upon a software component developed by Mark Shepherd using the Adobe
Flex development environment.149

Summary – 2003

The first year of this project was spent in developing several experimental works:
Memetic Variations, Jitter, Moebius and Nodelands. These experiments helped to clarify
the direction the project would take, and also identified two aspects that needed to be
resolved to further develop the research. The first of these was the medium of
expression: the nature of the programming language and the environment in which it
is used. Experience with Memetic Variations and Jitter had shown that C++ was not
suitable for open-ended exploration of a visual concept. The second problem was with
the algorithmic rendering of images with text - how to represent an image constructed
from a relatively small number of discrete elements with sufficient clarity to be
recognisable.

2004 – Assembling the Means

The year focused on exploring approaches to the technical and methodological issues
the previous year had raised. It was also an opportunity to look more widely and to
develop a greater insight into how other artists worked with software and generative
systems.

Expressive Programming Environments – 2004

I undertook an intensive survey of programming tools and environments to identify the
ones most suited for expressive programming: a programming environment where the
code is compact, comprehensible and powerful. My criteria included:

• A rich environment of predefined functionality and libraries of tools to draw
  upon, particularly for imaging.
• The need to have a simple but readable structure; be powerful, compact and
  concise, with not too many ways of doing the same thing.

149 Mark Shepherd, Springgraph Flex Component, Computer software, 2006. Available:
http://mark-shepherd.com/blog/springgraph-flex-component
Project Development

- Be able to doodle and sketch, that is, interactively explore an idea or expression by typing a few lines of code, without having to write an entire program.
- Have the ability to utilise graphic hardware acceleration.

I did not limit the selection to commercial or open-source tools and environments. The details of the analysis are contained in the next chapter, but the most promising candidate was a very high level language called Python.\textsuperscript{150}

NPR stands for Non Photorealistic Rendering - 2004

NPR is a field of research in computer science.\textsuperscript{151} Some of the results from NPR research can be found in Photoshop filters, which apply rendering effects to an image (such as the infamous plastic wrap filter). The core of NPR research is motivated by the recognition that detailed photographic representation is not necessarily the clearest way to communicate information through imagery. A schematic drawing of an engine disassembly is easier to understand than a photograph. NPR research is conducted in two distinct areas: in rendering geometric or 3D models (alternatives to ray tracing, such as toon and sketch shading); and by enhancing the presentation and comprehension of images.

NPR imaging research shares common problems with Computer Vision. A common problem is something that human artists find easy - how to separate a figure from its ground; how to determine the spatial relationships of objects in the image. NPR techniques often involve complex mathematics, and because they have direct


\textsuperscript{151} For an overview of NPR techniques see
Project Development

commercial applications, source code is not often made available by researchers. My intention in surveying the field was to glean techniques and insights that I could apply for rendering images with text. This wasn’t totally a waste of time, but I may have got more benefit from taking some drawing classes.

Previously the rendering of images had been by tone, either through the density of marks (number of pieces of text for an area), or through shading by changing the tone of the mark so that it was either adding or subtracting from the ground.

Some attempt was made to fit the mark to the image by aligning it along the image gradient. Drawn objects are often defined by their edges, so an obvious approach was to use edge extraction on the images.

Another observation was that if the image was drawn from back to front with progressively smaller text (with larger objects at the back) then better tone and finer detail should result.152

As part of my investigations into software and software environments I had come across an application called DrawBot.153 DrawBot is a simple programming environment for generating vector graphics with Python. I integrated this with the Vigra computer vision toolkit as a tool for exploring different image rendering approaches.154

152 The image Boys own camera club drawn using Appendix 2: Drawing Algorithm #5, 5000 words taken from,
Harvey G. Ralphson, The Boy Scout Camera Club, or, the Confession of a Photograph: Project Gutenberg, 2005.
http://toolshed.arthbt.utas.edu.au/billhart/phd/images/boysowncamerachub


Summary – 2004

There was little progress in the development of art works this year, but it was an important period of technical investigation which enabled the work to develop quite rapidly in subsequent years. A paper, *The Generative Arts and Novelty*, was written which explored the definitions of Generative Art and examined the practices of a number of artists working through the medium of software.¹⁵⁵

Selecting a programming environment is not as simple as making a decision about what software tool to use. For simple applications, a programmer can often hack or modify an existing piece of code without having a detailed understanding of the language they are using. For more complex applications, a deep understanding of any programming language requires a considerable investment of time to develop an understanding of the way problems are approached and solved.

Several analogies for learning a programming language can be made; the most common comes from the terminology *programming language*, which implies that programming languages are related to natural language. Learning a new natural language is more than learning lists of nouns and verbs, or learning the syntax and grammar; it involves learning to think in that language.

If however, the focus is on the functional aspects of programming - the process rather than the appearance - then a more appropriate analogy for learning a programming language can be found in children’s toy building systems, such as Lego, Kinex, Mecano, Mobilo, and Megablocks. Each of these systems has a syntax relating to how objects are connected, and this determines the types of structures that can be built. Learning to use the system requires experimentation to build up a range of techniques. Some systems have similar paradigms, for instance Lego and Megablocks are largely compatible. Lego is more complex; it has multiple paradigms, standard blocks and

¹⁵⁵ Parts of this paper have been incorporated into this document.

http://toolshed.arthbt.utas.edu.au/billhart/phd/generative_arts_and_novelty
Project Development

Technics. Each system has its own capabilities; it is far easier to build a roller coaster in Kinex than it is with Lego.\footnote{Visual programming metaphors, such as those used in MAX/MSP, jMAX and PD, make this explicit. In those software environments, the programmer is literally connecting objects together to create a machine.}

Selecting a programming language will influence what type of structures one can build and how effectively this can be done. Learning a programming language involves play-learning far more than memorisation.

2005 – Implementing the Methodology

This year brought a return to making, to the implementation and testing of the ideas and software technologies identified during the previous year.

Ocular Coincidences - 2005

Ocular Coincidences was a work I developed for Resonator, an exhibition curated by Derek Hart.\footnote{Derek Hart, Resonator, (Hobart: Salamanca Arts Centre, 2005).} It was a reworking of Memetic Variations, using Python as the programming language, and attempted to develop a stronger conceptual link between the imagery and the text.

Conceptualisation

The text for this work comes from Georges Bataille’s commentary upon the novella Story of the Eye in which he describes his childhood and the circumstances in which he wrote the novella.\footnote{Georges Bataille, Story of the Eye (New York: Urizen Books, 1977).}

Bataille had an amazing interdisciplinary talent - he drew from a wide range of influences and used diverse modes of discourse within his work. His novella The Story of the Eye, for example, published under the pseudonym Lord Auch was initially read as pure pornography, while interpretation of the work has gradually matured to reveal the considerable philosophical and emotional depth that is also characteristic of other writers who have been categorized within the literature of transgression.\footnote{Lord Auch, literally, Lord "to the shithouse" -- "auch" being slang for telling somebody off by sending them to the toilet}
imagery of the novel is built upon a series of metaphors, which, refer to philosophical constructs developed in his work: the eye, the egg, the sun, the earth, the testicle.

**Visual Manifestation**

I began by experimenting with *Verlet* integration techniques and simulated springs, as a way to animate pieces of text on screen.\(^{160}\) Springs are simple to simulate and have a range of controllable behaviours. The force on an object is directly proportional to the stretched length of the spring: if it is a soft or under-damped spring, then the object will oscillate around the resting location of the spring; if it is an over-damped spring it will slowly move towards the spring resting location. A tuned spring will return smoothly to its resting position. Rather than calculating splines to animate words between key frames, springs can be used instead with the advantage of not having to predetermine the sequence of key images. One shortcoming is that a single spring will only translate position, not alignment. By attaching two springs between each word and its key position, the words could be animated with the desired alignment. The programming transition from small-scale experimentation to animating a large number of words demonstrated the flexibility of *Python* for sketching. However, it immediately became apparent that, for intensive numerical computation, *Python* was much slower than *C++* (up to 100 times slower), and that a naive re-implementation of the original *C++* code wouldn’t be fast enough for real-time animation of the complexity that the work required.

A popular commercial numerical processing environment amongst scientists is called *Matlab*; its distinguishing feature is that it uses array arithmetic in a scripting

---

\(^{160}\) Developed by a French physicist Loup Verlet in the 1960s, the Verlet integration technique is far more stable and accurate than the Euler technique I had been using previously. The technique is presented in Appendix 2: Spring Simulation.
environment. This means that hundreds or thousands of numbers can be simultaneously operated on at similar speeds to a dedicated C++ program, but with only a few lines of code. A similar type of functionality has been added to Python through an extension module called Numeric. Animation of the words involved doing the same numerical calculation to each word every frame, so parallelising the program by using Numeric seemed an ideal solution. This did increase the frame rate substantially, but it was still below the minimal twenty frames per second threshold required for acceptably smooth animation. A profiling analysis of the program revealed that; it was the calls to the OpenGL functions to draw the words for each frame that were the source of the bottleneck. Until this problem was resolved, the approach of using Python for developing complex real-time animation was untenable.

Ocular Coincidences used a similar drawing algorithm to that used in Boys own camera Club, except that the number of words is reduced to 1900. To bypass the performance issues of the program, this work was rendered frame by frame as an

---

161 I had downloaded a demo version of Matlab, to experiment in finding ways to extract features from images.


162 Numeric has undergone a rapid evolution over the past five years. It started as Numeric, became NumArray, and both earlier versions have been combined to form the current NumPy. Various. NumPy. Computer software, 2005-2008. Available: http://numpy.scipy.org/

Perl also has a similar type of functionality with I had used with the GIMP image editor, called PDL (Perl Data Language).


These tools are developed and maintained by scientists, but offer a wealth of high performance, compact and low cost functionality to anybody interested in rapidly developing simulations. NumPy also includes sophisticated image processing, statistics and interpolation modules, and is part of a larger SciPy (Scientific Python package).

163 Profiling is a standard software analysis technique that accumulates the execution time for every procedure and function in the program, making it possible to identify where a program is spending the majority of its execution time.

In this case, four function calls are required to draw each word (position, rotation, scale and tone), making a total of 20,000 function calls per frame.

164 See Appendix 2: Drawing Algorithm #5

165 http://toolshed.arthbt.utas.edu.au/billhart/phd/mv/storyoftheeye
Project Development

animation that would play back from the hard drive of a G4 Macintosh tower. As there were no large flat panel displays available for the exhibition, an old installation technique was used. The computer and an old CRT monitor were installed inside a black plinth, and the image was displayed by reflecting the CRT monitor screen onto a sheet of acetate suspended above the box. This was a last minute solution, and there were problems with the rigidity of the suspension of the acetate. Unfortunately, after the work had been installed, the curator lit a wall behind the acetate which somewhat destroyed the effect.

Reflection

In focusing on learning to work with Python and resolving technical problems, I lost sight of the original conceptualisation I had for Memetic Variations. In trying to develop a coherent work for exhibition I had opted for a literal conceptualisation. The relationship between the source text and the imagery didn’t enhance the reading of the work.

Technically this was a step forward, but still clunky and awkward. The image formed was not particularly clear, and the timing of the animation was very slow. Feedback indicated that the most engaging aspect of the work was in the movement of the words, rather than the images formed, which appeared incidental.

Stories about You - 2005

Conceptualisation

Since my early twenties I have always tried to maintain an awareness of the different voices inside my head; the interweave of narratives constructing and retelling the stories that make up who we are. We believe we are one person, a single I, but we can be many people with different beliefs, capable of things we would never publicly admit. We, as conscious beings, are the sum of the stories that we tell ourselves about ourselves.
Project Development

The initial concept for this work came as an image of a head slowly being modelled by ribbons of text, on which partial narratives were slowly revealed to the viewer. Through the work of a student I had become interested in the phenomenon of confession web sites, and curious about the urge to publicly confess. I was also fascinated by the banality of most of the confessions, and the prurient satisfaction of finding a truly transgressive confession. My concept of this work, would be one that engaged the viewer in a public act of participating in the consumption and judgement of others, to be confronted by the shock of recognising the image being formed as themselves. The intention is to provoke a switch from spectator to subject, a greater awareness of self, and the multiplicity of others. We all have something to hide; we are all surprisingly alike.

Surveying the Field

The pioneer of the technologically mediated confession was The Apology Line, run by conceptual artist Allan Bridge from 1980 until his death in 1995. The Apology line was run through a telephone answering machine rather than a web site, and Bridge solicited confessions by advertising in local newspapers. Perhaps because the medium used was voice, rather than premeditated text, some of the Confessions Bridge obtained were extraordinary; death threats, confessions of murder, apologies for police brutality, the Vietnam vet who swapped his wedding ring for sex with a prostitute. Bridge released tapes and magazines from The Apology Line and the work was widely exhibited.


---

166 For instance http://www.grouphug.us or http://www.notproud.com

Project Development

I was initially confronted by the technical difficulties of this project:

- How to model a ribbon of text that was not a rigid string of works, but which floated and rippled, falling softly into place.
- How to detect and capture the images of gallery visitors faces.
- How to extract information from these images so that they could be drawn with the ribbons of text.

British duo Alison Craighead and Jon Thompson exhibited a work called *Decorative Newsfeeds*, which featured animated news headlines that sinuously moved across the screen, like snakes or worms.¹⁶⁸

The website of Israeli designer Ariel Malka contains a range of experimental works (particularly “The book of sand” and “Text time curvature”) made with *Processing* which provided some visual references.¹⁶⁹

I focused initially on the problem of simulating the dynamics of a ribbon of text. The starting point I chose was the real-time cloth modelling used in 3D games and graphics. I found quite readable source code for cloth simulation made by Jordan Isaak.¹⁷⁰ Isaak referred to the numerical methods used in a 2001 paper by Thomas Jakobsen entitled “Advanced

Project Development

*Character Physics*. This has become a significant paper for computer gaming and the simulation of soft and rigid body physics. It describes constraint-based stick and ball modelling (cloth dynamics), and equally significant for this project, it introduced me to the *Verlet* integration technique. The stick and ball constraint-based modelling is similar to the type of physics seen in the creatures on the website *Sodaplay*.

**Visual Manifestation**

I started with the kernel of the visual manifestation, the behaviour of the ribbon of text. Jordan Isaak’s code was written in C so it was fairly straightforward to modify. I then considered how to build a structure around this kernel of visual behaviour. I didn’t want to return to C++ based on my previous experience, and I knew that rewriting the ribbon code in *Python* would be too slow from previous experience. I had previously analysed the performance of the code and found that most of the execution time for *Python/OpenGL* programs was taken in accessing the *OpenGL* functions to position objects at each time step. It was the interface between *Python* and *OpenGL* that was the bottleneck. It is a programming maxim that 5% of the code in any application will consume 95% of the execution time, so the way to improve performance is to identify that 5% of code, and focus optimisation on that. *Python* is very expressive and compact for high-level data manipulation, but is slow for repetitive numerical calculation and foreign function access. Conversely, *C* and *C++* are very fast for numerical calculation and function calls.

The answer to my programming dilemma was to adopt a

---


Project Development

hybrid approach; do the majority of programming in Python except for the numeric and graphic kernels, which are programmed in C or C++. Python is built from the C programming language and has, as part of its design, functionality to access compiled C and C++ code. However, this is lengthy and tedious to set up for each function you wish to use. There are several tools for facilitating this functionality; I chose what appeared to be the easiest to use, SWIG.¹⁷³

This approach works well for reliable numerical code that doesn’t require debugging, and the hybrid application runs at almost the same speed as the pure C or C++ application, but is more compact and malleable.

Content Selection and Data Preparation

Content for this version came from the website http://www.notproud.com; the stories were extracted by a Python script which uses built-in modules to retrieve text from the website.

Finding a means to detect and capture an image containing a face from a video camera turned out to be remarkably easy. I found that the OpenCV (Open Computer Vision) library had a sample face detection application, which appeared robust enough for my needs.¹⁷⁴

Once the face had been detected and extracted from the video image, it was converted to a drawing using edge detection software from the Vigra library.¹⁷⁵ The edges were then broken down into a series of points and fed to the animation engine via a data file. As the ribbons were created, they were assigned

¹⁷³ Using SWIG involves writing an interface definition file that can be as simple as 5 lines of code. The SWIG application then analyses and generates the appropriate interface code.


Project Development

specific points to be attracted towards as they fell, outlining the edges of the drawing.

Reflection
I was very pleased that I now had a viable methodology for constructing this work.\textsuperscript{176}

I was impressed at how easy it was to add on the high-end functionality of face detection, and the ease of integrating the C++ code for simulating the ribbon and the Python code for adding the text texture and controlling the release of the ribbons.

However the work was still some way from being ready for installation; there were a number of unresolved aspects:

- The stories were quite long, and at times, overly confronting; the thread of the story could be lost, and the content seemed to be too intently shocking.
- It took a long time for the face to form (around 40 minutes).
- The text ribbons were transparent, but because of the way OpenGL renders transparency, the transparency wasn’t always visually correct, which could make it difficult to look at.\textsuperscript{177}
- It was difficult to control the level of detail in the final image.
- There were installation considerations about how to unobtrusively install a web camera to detect and capture faces.

Summary – 2005

The technical progress this year had been pleasing and I felt I was on the right track. I had found a programming medium that was both powerful and expressive and had found solutions to software performance issues. There were a number of aspects to the visual appearance that I wanted to address, including transparency, shadows and anti-aliasing.

Conceptually, the work hadn’t advanced that much though; none of the works were close to the resolution I would want for an exhibited work. The understanding that comes through making still seemed some way off.

\textsuperscript{176} http://toolshed.artschool.utas.edu.au/billhart/phd/stories/storiesv1

\textsuperscript{177} For transparency to work properly, the non-transparent polygons must be rendered first, and then the transparent polygons must be sorted by depth and rendered from back to front. I was amazed when I realised this functionality wasn’t built in.
2006 – Putting it into Practice

I was fortunate enough to have the first half of 2006 to devote full time to research. Again the focus of this year was largely technical, finding software solutions and learning the nuances of this software environment. It also saw a considerable fecundity in generating new ideas for works and, through a developing fluency with writing the software, the ability to rapidly prototype new works.

Scene Graphs – 2006

I started the year by trying to implement proper transparency and shadows through OpenGL. I quickly realised that this was something I didn’t want to do. While there were numerous samples and examples of how to do it available, it would be very time consuming and I would end up building a software engine. I realised that I had been trying to use the functionality of OpenGL at the wrong level.178 My initial reason for choosing to work with OpenGL and not with higher-level software was that I didn’t want the design decisions of a game engine to bias the visual appearance of my work. I undertook a survey of available tools (see Chapter 5 for details) and eventually understood that I needed a Scene Graph; the one I chose to use was OpenSceneGraph (OSG), which had a Python interface PyOSG, although this was later to prove problematic.179,180,181,182

---

178 Software, like physics or biology can be thought of as having different levels of description. The binary machine code is the lowest level, followed by assembler code, languages like C etc. Software functionality is similar: different API’s (Application Programming Interfaces) have different levels of functionality.

179 A Scene graph is a data structure used to organise and display geometric objects.


182 OpenSceneGraph was developing rapidly towards its release as version 1.0. In the run up to this milestone release, features would change between minor versions as the architects of the software refined their ideas. Gideon May, who had originally written PyOSG, had disappeared. With some assistance from Joseph Winton, I updated parts of the software to be compatible with the 1.0 release version of OSG. PyOSG had been developed using a different type of wrapping software called boost.python, which required a much deeper knowledge of
Memetic Variations (Ontological Drift) – 2006

Conceptualisation

In reflecting upon this work after Ocular Coincidences I had a realisation that I couldn’t represent my intention through one screen; rather, it would require several (four seemed the right number) to explore different aspects of the transformation from knowledge to knowing being. Memetic Variations came to stand for two things: the name for the suite of works that I eventually titled The Conditions for Ambient Cognition, and the specific work that became Ontological Drift.

My concept for the whole work, in a not yet clearly articulated form, was to represent a cyclic process of movement between language, perception, thought and action – a movement from knowledge to understanding, communication to perception, abstraction to physical action.

In returning to this specific work I wanted to make a connection between an awareness of being and the frameworks of knowledge and belief through which we perceive the world. Systems of knowledge and belief influence the way we view and engage with our environment but being is, in essence, the awareness of a series of perceptions of now.

I chose two lists of words from Wikipedia, one of words ending in -ism, and one in words ending in -ology, as being representative of belief and knowledge. These words would be animated to form the images that suggested being in the world: snapshots, fragments of perception, memories.

C++ than I possessed. A characteristic of boost is that it can generate compiler errors that go on for hundreds of lines. I developed a real hatred for boost.python, and was very relieved when Paul Melis privately communicated to me his swig-based python wrappings for OSG – PyRand in late 2006, which have been much easier to maintain.
Visual Manifestation

This was the first work I implemented using OpenSceneGraph, modifying the engine I had developed for Ocular Coincidences. Aside from some initial performance issues that arose in learning to use the scene graph software, this worked well. Visually, because words ending in -ism and -ology tend to be quite long, the animation tended to look rather spiky and awkward.\(^{183}\)

I thought that if, instead of being straight, the text curved to conform to the contours of the image then it should form a clearer image. I developed a new drawing algorithm that found arcs of constant tone on an image.\(^ {184}\) However, rendering the curved geometry of the arc strips places greater demands on the hardware graphic processor and, consequently, I had to reduce the number of words being animated to 2000.

Reflection

I felt that this work was becoming both more visually and conceptually resolved, but there were still a number of issues to be addressed.\(^ {185}\) Using curved text required more graphics processing power, so I had to use less words; this tended to give the imagery a contoured appearance. Because I had thought there were likely to be artefacts near the edges of images, I had vignetted

\(^{183}\) [http://toolshed.arthbt.utas.edu.au/billhart/phd/mv/meme3](http://toolshed.arthbt.utas.edu.au/billhart/phd/mv/meme3)

\(^{184}\) See Appendix 2: Drawing Algorithm #6

\(^{185}\) [http://toolshed.arthbt.utas.edu.au/billhart/phd/mv/meme4](http://toolshed.arthbt.utas.edu.au/billhart/phd/mv/meme4)
Project Development

most of the source imagery. The vignetting tended to dominate the contouring. The final resolution of this work would require a more powerful graphics processor and careful selection of source imagery.

*Quote (Dialectic Seepage) – 2006*

This is one of the few works I have made that has just worked, without needing to be left, reflected on and reworked.186 The earliest version is as described in Chapter 2. I had met Dr Sally Pryor at the University of Melbourne and she introduced me to *Integrational Linguistics*; I was immediately struck by how many of the core concepts of *Integrational Linguistics* related to my own developing thoughts on language and communication. The inspiration for this work came from grappling with linguistic concepts, particularly the interpretation of codes and meanings in different linguistic theories. The work was shown as *Quote*, in an exhibition curated by Dr Colin Langridge entitled *Quote*.187

*Jitter (Transcendental Jitters) – 2006*

**Conceptualisation**

This work was a continuation of the concept from the work of 2003 with the same name. The intention was to create a relationship between the physical detritus of the body and the transcendent platonic realm of ideas; that through scrabbling in the dirt, an abstract objective truth about the world can emerge. In particular, I was thinking about the equations of physics which expressed the relationship between different types of physical quantities such as those for mass, energy, electricity, magnetism, force and gravity.

---

186 http://toolshed.arthbt.utas.edu.au/billhart/phd/mv/quote

Project Development

As a dynamic metaphor I decided to use Brownian motion, which is the random motion of particles under the influence of thermal buffeting from unseen molecules; which results in a characteristic jittery appearance. Brownian motion is quite easy to simulate through random number distributions.\(^{188}\) In similar fashion to the 2003 version, an image would be formed by imposing a source image as a field that influences the movement of particles, by increasing friction where the source image differs from a ground.

**Visual Manifestation and Data Preparation**

For this project I collected and photographed nail clippings to use as the particles to animate, and scanned a range of formulas from my first year university physics textbook from which the images would be formed.\(^{189}\) Nail clippings are surprisingly difficult to photograph without shadows, so I constructed a camera stand to photograph the clippings on glass, lit from the side so that the shadows were separated from the objects.\(^{190}\) I made extensive use of the *Numarray* random number module for *Python* for generating random numbers with Gaussian and Poisson distributions.\(^{191}\) The work had a coloured background, an image of graph paper, which was the first time I had used colour in this medium.

---

\(^{188}\) The algorithm used is based on a sample program, *Brownian.py*, that comes with the Python source distribution. It is unattributed and undated. See Appendix 2: Brownian Motion Simulation


\(^{190}\) I also used the same system for photographing the objects for *Classic*.

\(^{191}\) A Gaussian distribution has the characteristic bell curve shape.
Project Development

Reflection
Again, construction of this work was straightforward. I was satisfied with the movement of the nail clippings, but needed a greater diversity of samples.192 However, the formulas were only readable if they had a few terms, and I realised that the references would be likely to be misread as an obscure code by an audience who were unfamiliar with the reference.

Classic (Faith in Reason) – 2006
Conceptualisation
This piece is a return to the animation sketch *Words* I had created using the *GIMP* prior to starting this project. The conceptual image was of a relationship between symbolic thought and physical action. Tools are implements that act upon the world and enhance our ability to influence our environment. To develop tools requires abstract thinking, an understanding of the principles through which the tool will act.

Classical determinism is based on the belief that, if the guiding principles of matter are known, and there are precise measurements of where everything begins, then you can in theory predict the future. The motion in this work is a reflection of this; it is based on cubic splines, so motion paths are predetermined and fluid, in contrast to the unpredictable random movements of *Jitter*.

192 http://toolshed.arthbt.utas.edu.au/billhart/phd/mv/jitter2
**Project Development**

**Visual Manifestation and Data Preparation**

For this work I photographed a collection of small useful objects and tools in my collection to form the key images, chosen for their definite shape as this work uses a drawing algorithm that renders form rather than tone.\(^{193}\) I collected a number of freely available math fonts; the open-source font editor *FontForge* was used to extract suitable symbols and numbers.\(^{194}\)

**Reflection**

This work has a smooth-flowing, almost three-dimensional appearance, as the symbols move from one shape to another.\(^{195}\) The key images appear to snap into focus and hang for a moment before transitioning to the next form. The parametric interpolation between key images is nonlinear; the motion is slower near the keys and faster in-between. The background texture of paper helped to contextualise the image as a human cognitive space, but the variety and kind of mechanically generated mathematical symbols didn’t resonate with human cognition.

**Exploration Readings - 2006**

**Conceptualisation**

I had long wanted to create a work using *Project Gutenberg*, a repository of non-copyrighted texts that have been converted to electronic format by volunteers.\(^{196}\) I also wanted to revisit the engine of interacting pages I had used in *The Writing*

---

\(^{193}\) See Appendix 2: Drawing Algorithm #7

\(^{194}\) Math fonts are freely available from the *TeX* typesetting system and from *Mathematica* software.


\(^{195}\) http://toolshed.arhbt.utas.edu.au/billhart/phd/mv/classic/play

Project Development

*Machine* and envisaged an installation where any movement in front of the screen would cause the pages to ripple and interact.

**Surveying the Field**
From the cut-up techniques of Tristan Tzara through to William Burroughs and Brion Gysin to more contemporary times, generative processes for generating chance meanings has been a theme of twentieth century art and literature. Examples of recent reading machines include Ken Perlin’s *Kinetic Poetry*, which dynamically generates readings in a web browser through a jostling simulation of fridge-magnet like words.¹⁹⁷ *Screen* is a virtual reality work that literally immerses a viewer in bodily interaction with words.¹⁹⁸

**Content Selection**
I chose to use e-texts from Antarctic exploration journals from the Australian *Project Gutenberg* archive as a starting point for this work. Random pages from the e-texts are rendered with a handwriting font onto an image of paper.

**Reflection**
The animation engine is based on stiff cloth dynamics, which worked far better than they had for *The Writing*

Project Development

*Machine.* However, this work was not driven by a clear conceptual image, and didn’t appear to fit well with the rest of the work as it was developing.

**Stories about You – 2006**

**Conceptualisation**

The previous works had all been developed for orthographic projection, whereas *Stories about You* used a perspectival projection. I wanted to include lighting, more complex materials and shadows to improve the modelling of the ribbons of text as objects in space; the shadows would add a dual reading of the text as it floated through the space.

**Visual Manifestation**

A number of times I have been surprised by how badly 3D graphics approximates the physical world. In the physical world, shadows are a direct consequence of lighting; in 3D environments they have to be approximated. There are two main techniques for generating shadows (and numerous variations and improvements): using geometry to calculate a shadow volume, or calculating a shadow map for each light source to superimpose on the rendered scene. I chose to use a shadow map technique, as shadow volumes aren’t able to cast an accurate shadow from a texture with transparency. Starting from an OpenSceneGraph shadow example this was straightforward to implement, but the shadows were very

---

199 http://toolshed.arthbt.utas.edu.au/billhart/phd/exploration-reader

200 In orthographic projection objects do not scale with distance. Orthographic projections are ideal for simulating a two dimensional view.
Project Development

pixelated and prone to z-fighting. This was resolved through brute force by using as large a shadow map as possible and adding a fragment shader to blur the shadows.

Reflection

This was the first work in which colour had become overt. For many years I had avoided using colour explicitly, initially because the colour space on computer screens was restricted and highly variable between different display devices, and later from habit. While I was now happy with the appearance of the work and the way the text was introduced to the screen, there was no mechanism as yet to clear the drawing and start the next. I was not particularly satisfied at the time with the appearance of the drawing and how long it took to draw. The stories from http://www.notproud.com had begun to seem overly long and overtly provocative.

In practical terms, while I had the parts of the installation functioning, there were still difficulties with how the work would be installed. I did a test installation of a web cam in a public doorway to capture faces, and found that in most cases the images were unusable. People needed a reason to pause and look at the camera long enough to be detected and a clear image obtained.

Words of Art (The Transient Taxonomies of Art) – 2006

Conceptualisation

I had been considering ways of making pictures as portals into simulated environments for a number of years. My initial conception had been of landscapes in which external environmental factors would influence the simulated weather and lighting. A conceptual starting point for this project was Richard Dawkins’ concept of the Meme; I was intrigued by the concept of ideas having a life of their own, and so I conceived of simulating an idea ecology. Ideas would arbitrarily cluster together to form unstable Meme complexes. Ideas would become tired and die, some would become hugely important for a short while, and some would consume other ideas.

---

201 Z-fighting is a technical term for an irritating flickering in parts of the image, due to insufficient numerical accuracy in calculating the distance of objects from the viewpoint.

202 A small program that runs on the graphic processing unit and manipulates pixels.

203 http://toolshed.arthbt.utas.edu.au/billhart/phd/stories/storiesv2
Project Development

Surveying the Field

The simulated aquarium is one of the most enduring and popular screensavers; similarly, there are also screensavers available for waterfalls, gardens, storms and butterflies amongst others.\textsuperscript{204} Clearly, the idea of the screen as a portal to a simulated ecosystem has a popular appeal. While most of these screensavers depict an environment too perfect and lush to be real, and eventually too predictable to be interesting, the idea of simulating environments in which things evolve and interact has allure.

Artificial Life and evolutionary simulation has been an active area of research for artists since the late 1980s: Karl Sims, Jon McCormack, Christa Sommerer and Laurent Mignonneau were some of the pioneers. My intention was closer to that of the screensaver than most A-life art. I wanted to examine the idea of memes, making them explicit visible things. Memes are an alluring concept, but do they have an objective reality? Do they tell us anything about ourselves? Or will they be another idea, like the luminiferous aether that future generations look back on with smug superiority?

Like many A-life systems, my simulation would be an agent-based one, similar to those implemented with the software Breve.\textsuperscript{205} I eventually decided that a modified flocking algorithm with predators and prey would form the basis of the interaction of the agents. The flocking algorithm was developed

\textsuperscript{204} A quick search on Download.com reveals 90 different aquarium screensavers available for download as of June 2008.

Project Development

in 1986 by Craig Reynolds, and has become popular as a demonstration of emergent behaviour.\(^{206}\)

In looking for easy ways to implement flocking with *OpenSceneGraph* I came across *Crogai* by Nicolas Brodu, a library that uses Craig Reynolds’ *steering behaviours* for the simulation of crowd behaviours.\(^{207}\)

**Content Selection and Data Preparation**

In thinking about force-directed graphs, I had come across the glossary website *Words of Art* by Robert Belton, which I had thought would make a good knowledge domain to visualise.\(^{208}\) I decided to treat it as my meme pool, a collection of related ideas. It had the advantage of being not just a collection of terms but also, through hypertext linkages, a collection of related ideas.

I had also been looking at *Nodebox* research, in particular their *Prism* module that used Google as a repository of knowledge about the world to derive a colour palette for a word.\(^{209}\) I was struck by the profundity of this: as the web grows and search engine spiders index it, then the indexes must contain a vast accumulated knowledge about the world; the ingredient that has always been missing from strong A.I.

I wrote a program to extract the terms, and relationships between terms, from *Words of Art*, and then used the Yahoo search engine to determine the significance of each term.\(^{210}\) A colour was assigned to each term by finding the most dominant hue and tonal associations with the word across multiple searches, and from these, a weighted

\(^{206}\) Reynolds. "Flocks, Herds and Schools: A Distributed Behavioural Model".


\(^{208}\) Belton, *Words of Art*

\(^{209}\) De Smedt and De Bleser, *Prism*

\(^{210}\) Yahoo lets you do 2000 searches per day whereas Google only allows 1000.
Project Development

average was calculated. This took several days to execute for the four hundred terms.211

Visual Manifestation

I had imagined my memes as words swimming through the aether, so I modified my ribbon code from *Stories about You* to be simpler and to have rhythmic spermatozoa-like swimming motion.

Nicolas Brodu’s software already had a sample application demonstrating flocking behaviour: to convert this to *Python* and wrap his library seemed like a lot of work, so I grafted my ribbon code onto his agents and added an interface to his software so that it would load the data I had collected via *Python*.

Reflection

I saw this as a first sketch of this work, and it seemed to have some of the qualities I wanted. The next stage would be to add in some form of energy / significance interaction between agents, mechanisms to gain energy and importance through predation or clustering.

Summary – 2006

At the conclusion of this year I felt that I had made real progress towards creating a group of resolved artworks. I had had considerable technical frustration with building and porting the PyOSG code to different versions of OpenSceneGraph and computer operating systems. However, when the code was installed and working, constructing and refining a work was productive and fun. Towards the end of 2006 I contacted a user of OpenSceneGraph, Paul Melis, who provided me with a copy of his unreleased

---

211 This was dumb AI. How can an art term like *deconstruction* have a colour?
Project Development

swig wrappers for OpenSceneGraph, which are much easier to maintain, and I have been using them since then.

Although the work was starting to come into a conceptual focus, this was not yet sharp or clearly articulated, and although I was interested in the views of communication and particularly writing of Integrational Linguistics, these were not something I was trying to impose on the work.

2007 – Reflection and Refinement

The first part of this year was spent in reworking Words of Art, removing the dependence on the Crogai code and developing my own flocking implementation. I acquired four identical 22" LCD monitors and began to give serious consideration to the configuration and installation of multi-panel works. The studio practice changed during this year. In the second half of the year it was no longer driven by technical problems, but moved to a reflective and intuitive engagement with the work. Time was spent watching, tweaking parameters and media to clarify the meaning and appearance of the work, rather than fixing bugs and coding.

Words of Art – 2007

Visual Manifestation

I came to the realisation that I had reached a dead end with the development strategy I had used for Words of Art in 2006. Nicolas Brodu’s software, which formed the basis of the project, had compatibility problems with newer versions of OpenSceneGraph and was more generalised and complex than I needed. I was having difficulty in finding ways into the software to customise it further. Embedding Python code into a C++ program lost a lot of the advantages of introspection and flexibility that I had had with the other works.

I decided to implement a flocking algorithm from first principles using Craig Reynolds’ original paper. The flocking algorithm implementation was done initially with matrix algebra using NumPy, and then later converted to C++ for performance. This was, in retrospect, perverse: there are numerous sample implementations, and mine was a novel implementation based on force and acceleration rather than on velocity, which caused additional problems.
Project Development

Although this rewriting took quite some time, I did gain a better understanding of the dynamic possibilities of the system by the time it was completed.

Reflection

At the end of the process of rebuilding this work, I had developed the work to the same level as the sketch from 2006, except that now it was my implementation, and the program was largely written in Python. It could now be displayed across multiple screens, and I was ready to implement the energy/significance exchange system between the agents.

Stories about You – 2007

It felt time to resolve this work to the stage where it could be installed and exhibited. The dynamics engine was tuned so that the ribbons of text entered from the right, fell to the drawing surface, and when the drawing was completed, blew off to the left. The drawing process was now much faster (ten to fifteen minutes to do a drawing, as opposed to forty).

I was dissatisfied with the length and sensationalism of the stories from NotProud and, during 2007, I found the website http://www.onesentence.org which has a moderated collection of true stories told in one sentence. I wrote a script to download and collate these stories and then edited them to remove or alter overt references to personal gender and location.
I had not been satisfied with the clarity and consistency of the drawings, which were based on a standard edge-detection process, and tended to be either too complex or sparse. I came across a computer vision technique called *Active Appearance Modelling*, which enabled line drawing caricatures of faces to be drawn from video imagery, exactly what I wanted.\(^\text{212}\) Unfortunately, my searches to find the core components of software to implement this system were fruitless, and I gave up on this approach due to the time it was likely to take to implement.

My pragmatic approach was to re-implement the face detection and drawing software using the *OpenCV* library and to develop an iterative process to adjust the sensitivity of the edge detection until it had an image with the required number of lines.\(^\text{213}\) I then developed a simple server to capture faces, process them and then, on request, to provide the most recent face that had not yet been drawn across a network.

My final task for this work was to find a way to get people to pause and look at the camera; the next and final work was in response to this problem.

*Flow (Communal Sense) – 2007*

**Conceptualisation**

I had been using the built in *iSight* camera in my laptop to

---


develop the face detection and drawing software when it occurred to me that people will look at the camera if they have something to look at, and that all new iMac's have an iSight camera built into the screen.

I had been interested in doing a work that explored the intentionality of reading, which would require an act of intention to be able to read. I was also interested in exploring optical flow, a computer-vision technique, which determined the flow vectors of pixels from successive images, rather than just the differences. This version is as described in Chapter 2, Communal Sense.

Visual Manifestation
Initially, I constructed the work without the interaction generated from optical flow. I used the text layout and animation engine from Quote and started to perceive the work as a kind of Socratic dialogue. The two figures are rendered with text from all the quotations, and the words for each quotation arise equally from their places on both figures and join to form the quotation as it approaches the viewpoint of the viewer.

The computer that would be displaying this work needed to be doing several things at once:

- Monitor the video camera:
  - Calculate optical flow for each frame.
  - Detect faces; if one was found, then calculate the drawing for that face.
- Make the drawing data available across the network on request.
- Run the animation software.

Modern iMac computers have two processors so, in theory, they can undertake these tasks simultaneously. Unfortunately, a design limitation of Python means that a single program can only use one processor at a time.\(^{214}\) I resolved this by having a master program that started three other programs: one as a server, one to watch the camera

\(^{214}\) The most common procedure for doing several things at once with software is called threading. With languages like C++, threading can occur over multiple processors. Unfortunately, Python had a thing called the GIL (Global Interpreter Lock) that only allows one Python thread to run at a time.
and calculate optical flow, and the other to execute the animation. The animation
and camera process could communicate data via an area of shared memory.

**Reflection**

I enjoyed making this work; it was personally meaningful, and very
satisfying to resolve the installation problems of *Stories about You.*\(^{215}\) It
was also a great example of how every time I’d gone looking for something with *Python,* I’d been
able to find a ready-made solution. After confronting the GIL
problem, I tried several
multiprocessing packages before hitting on the idea of using shared memory to allow
different processes to exchange data, and finding a module to do just that within
minutes.\(^{216}\)

Optical flow gives a sense that the words can be pushed around, and as the words
come closer to the viewer, they become more sensitive and difficult to control. Like
most reactive gallery works, once the viewer realises that the work is responding, their
engagement can be reduced to just exploring the boundaries of interactivity, rather
than the work’s intention of encouraging stillness.

**Summary - Tweaking and Resolution – 2007**

2007 ended with a sense that the project was entering its final phase; the
technological obstacles had been overcome, and attention could be turned to the
functioning of the works on an aesthetic and perceptual level. The intellectual
engagement with the work had changed from analytic problem solving to its sensory
appearance and reception. This entailed resolving the introduction of colour to all

\(^{215}\) [http://toolshed.arthbt.utas.edu.au/billhart/phd/flow/flowv1]

\(^{216}\) Admittedly, you have to know what you are looking for, and a long-term experience with Unix
systems makes that easier.
Project Development

the works of The Conditions for Ambient Cognition, selecting and substituting hand-drawn symbols for Faith in Reason, and hand-written words instead of formulae for Transcendental Jitters. The addition of colour greatly helped the perception of imagery for Ontological Drift.

The next step was to contextualise the intuitions that the work embodied and to consider the visual manifestation of the work. I began thinking about the gallery installation of the works; I wanted to remove the screens from their contexts as computer monitors and mass-produced consumer electronics, and so constructed a sample frame in which to house a monitor.

Words of Art (The Transient Taxonomies of Art) – 2008

*Words of Art* was the final work to be resolved, and is the work with which I am least satisfied. I refined the swimming movement of the wrigglers and implemented the predatory / prey and energy / importance transfer system. On reflection, I was not sure how much it added to the work. It occurred to me that my ideas had developed over the course of this project and, with this work, I had been engaging in an unconscious irony with myself. This became clearer when I came across *Babelswarm*.

*Babelswarm* is a recent artwork constructed in the virtual space of *Second Life*. The premise for *Babelswarm* comes from the myth of the Tower of Babel where, for the hubris of attempting to build a tower to heaven, God took away the language of Adam (the language to the true names of things) and left the builders without a common tongue.

In *Babelswarm*, visitors to the second life installation participate in building a tower of Babel through the words of their avatar.

---

217 Christopher Dodds, Adam Nash and Justin Clemens, *Babelswarm*, 2008.

218 *Second Life* is a Multi User Virtual Environment, by Linden Labs (a privately owned American company). *Babelswarm* was funded by an Australia Council residency grant.
Words they type (at the prompting of a disembodied entity, if no one else is there) fall out of the sky as cubes, which break apart into letters and jostle to find a place in the existing structure. I was initially intrigued by the aptness and the poetry of the metaphor, and the installation is undoubtedly a technical achievement. I was left wanting more, and a radio interview with the artists I heard helped me place what was lacking.\textsuperscript{219} During the interview one of the artists described how the use of swarm theory meant that the structures of letters the work builds are unique and unpredictable.

But I found the reality was less than satisfactory: you can see the blocks fall from space, and they are visually impressive for their number and translucent quality. The blocks jostle into place, but the structures formed appear little more than random, a structure caught in the process of falling down rather than building up. The promise implicit in the title, of a structure that builds from the fragmentation of language, was not evident.

In further reflection on \textit{Words of Art}, I think that it suffers from the same syndrome. Beyond the initial impression of the flocking words, and the chance association of words, the work does not continue to develop in complexity.

In this I may have a case of sea-monkey syndrome, an expectation that this new technology should do something more.\textsuperscript{220} Millions of people have been content to watch fish being nothing more than fish in aquariums.

\textbf{Trial Installation – 2008}

Installing a computer running custom software in a gallery context is asking for trouble; invariably the viewer is confronted with the blank screen of a computer that hasn’t been turned on properly or has crashed. Software in the gallery is not the same as in the studio; installed computers run for long periods of time and can become unreliable with heat or cold or fluctuating power supplies. It is impossible to certify

\textsuperscript{219} Amanda Smith, \textit{Artworks}, ABC Radio National, 2008.

\textsuperscript{220} Sea Monkeys are brine shrimp. The larvae can survive in a desiccated state and come alive when water is added. Comics of the 60s and 70s had advertisements for Sea Monkeys as humanoid creatures living as families in underwater houses and palaces. Millions of innocent minds were disillusioned.
Project Development

that any piece of software is bug free. The smart thing to do in gallery installation is to create a movie from the software and play the movie on a loop. The gallery viewers will never know.

The work for this project is installed as software, because:

- One piece is reactive to the viewer.
- One piece requires real time data.
- One piece plays across multiple monitors.
- Generating the imagery in real time allows for high resolution and images free from compression artefacts.

As a trial for the submission installation I installed all the works in March 2008. After several days of stress testing each work in the studio, and running faultlessly for several days in the gallery, several works froze ten minutes into a critique with thirty people in the space – a salutary reminder of the difficulties of working with this medium!

This was the first time I had seen all four parts of The Conditions for Ambient Cognition together, and it was vital for informing the final decisions to be made for the work.

Summary

The studio work progressed through five distinct phases:

1. Exploring the Possibilities – short experimental works that identified gaps in knowledge and capability, and provided markers through which to locate the project.

2. Assembling the Means – surveying and evaluating potential solutions to problems identified previously.
3. **Implementing the Methodology** – trying out the identified solutions on work previously implemented with other systems, identifying and resolving technical obstacles. The analytic focus required means that any outcomes are likely to be a demonstration rather than a conceptual or aesthetic resolution.

4. **Putting it into Practice** – the technological obstacles overcome, there can be a rapid development from concept through to prototype software.

5. **Reflection and Refinement** – the processes of programming (analytical thinking) requires that the original fuzzy conceptual image becomes a set of concrete objectives. The consequent aesthetic decisions can be poor. This stage involves a reconnection with the conceptual image, and evaluation and reworking of the artwork in the light of it.

Each phase requires a different mix of cognitive modes and attributes. The three significant cognitive modes I identify as:

- **Thinking through words** – locating, evaluating, assimilating and synthesising knowledge through the written word. This is characterised by the ability to develop deeper understanding by navigating the rich web of associations words have.

- **Thinking analytically with symbols** – an assertion that comes from this project is that programming has no relation to written or verbal language. Thinking through programming is more analogous to other types of symbolic reasoning such as musical notation or mathematical proofs.

- **Thinking with image concepts** – this is non-verbal intuitive thinking that explicitly involves sensation and a somatic engagement with the idea. More than just visual appearance, this also involves the experience of time and movement. Concepts form as sensation images, and thinking is often felt through the body – a sense of rightness or wrongness to a decision, for instance.

The challenge for the software artist is to be able to function creatively in these very different cognitive modes. I personally find it takes some time and effort to achieve a deep engagement with each mode, and it is particularly crucial for the software artist to be able to switch from analytical thinking to thinking through image concepts. To engage in writing software the mind needs to hold a mental model of the program, its
Project Development

detailed form and structure. To create an artwork, the mind needs to perceive the correspondences and connections between the internal conceptual image and the work.

Conclusions

The methodology developed through the process of making the works described in this chapter is claimed as a research outcome of this project. It expected that the explicit software methods of combining high-level programming languages with embedded low-level language components would currently only be useful to artists with a solid technical grounding. It is acknowledged that artists such as Ben Fry and Gideon May have used similar hybrid techniques previously, but I have not found a description of this integrated methodology.221

The current focus in software art is based around software environments such as Processing, Flash, MAX/MSP, Nodebox, which allow for the development of a community of users and a knowledge base of experience. However, it is claimed that the approach developed here is more open-ended. Sophisticated development tools and practices can be used, such as integrated development environments with debuggers and code version management. Potentially, any existing software can be integrated into an artwork; the creative environment is not just the package, or the operating system, but the vast collection of tools and techniques available on the global communication network. This approach meets the four key criteria for software art: introspection, expressiveness, flexibility and extensibility.222

221 Gideon May developed the first Python OpenSceneGraph wrapping, and worked with Bill Seaman on the Bill Seaman and Gideon May, The World Generator, 1997. and Ben Fry uses Perl as a data preparation tool see Fry, Valance.

222 Introspection is the ability for software objects to tell you what they are, what they can do, and how to use them.
Chapter 5 – The Materials of Software Art

This chapter presents analysis and observations about the materials of software art that have arisen over the course of the project. In particular it focuses on:

- *Programming Language as a Material* – the concept of expressive programming is introduced and its conditions defined. A rationale for the choice of language used in this project is presented and compared to other current options. This section finishes with a discussion about levels of abstraction in software and their influence on expression.

- *Drawing in 3 Dimensions* – the choice of software to visually express the project is described.

- *The Materiality of Numbers* – this chapter finishes with three observations on the material properties of computation in relation to software art: Software as a score in relation to its performance; The two strategies of media representation sampling and synthesis; Pseudo randomness as a material quality.

**Programming Language as a Material**

I am fortunate to be old enough to have dabbled with generations of programming environments ranging from entering machine code by key switches, assembler coding numerically controlled machines, through to writing parallel software for super computers. During that time I have worked with a wide variety of programming environments and scripting software. I don’t claim to be a particularly good programmer; my emphasis has always been on achieving a result, not upon how elegantly it is done. Programming languages all have design philosophies and zealous adherents to those philosophies. The sheer number of programming languages in existence (8500+) is testament that there is no perfect programming language.\(^\text{223}\)

\(^{223}\) Pigott, *Hopl: An Interactive Roster of Programming Languages*
The Materials of Software Art

Expressive Programming

The pragmatic choice of programming environment is dependent on both the abilities and experience of the programmer and the intended application. As the application for this project was to create artworks, the ideal development environment is one that has both the technical capabilities to make the work, and to facilitate the development and refinement of the ideas underpinning those works. This expressiveness relates to the compactness of the code (how quickly and succinctly it can be written), its comprehensibility (ability to interpret the structure and function of code once written), and its power (ability to represent a diverse range of structures and formal concepts). The programming environment acts as an interface between the programmer and the cognitive extension of the computer. The more quickly and succinctly concepts can be translated to code, the more transparent and expressive this interface becomes.

Desirable characteristics for expressive programming:

• Language should be *interpreted* not compiled (but ideally both). Interpreted Languages have an interactive interpreter that allows experimentation with ideas and expression through the immediate evaluation of short pieces of code as they are entered.

• Language should be *introspective*. Introspection allows the programming environment to be interrogated. Introspection means the programming environment can answer questions like: what are you? what can you do? how do I use you?

• Language should be *readable*. Writing a piece of software, and then later trying to reinterpret what it means, are two different things. Languages that encourage readable code make it easier to comprehend a program’s function. An aspect that aids comprehension is if the language is not too flexible in its expression, that is, there are only a small number of ways to do most tasks.

• Language should do *housekeeping*. Automatic memory management and declarations; some would argue these are the antithesis of good programming, but not having to declare variables and allocate memory helps with free expression.
The Materials of Software Art

- **Language should be rich.** The language should provide a rich environment of tools for common tasks like manipulating text and data, networking, imaging and handling interaction.
- **Language should be ubiquitous.** The language should be available on a wide range of computing environments, and should ideally come with the operating system.
- **Language should have friends.** Having a community of active users means that you will be able to find answers to problems; it also means it’s likely to be around for longer.
- **Language should be promiscuous.** The ability to interface with other languages and software libraries means never having to port code or be stuck with the second best library.
- **Language must have an IDE.** An Integrated Development Environment is a productivity tool that lets the user manage projects, edit code, manage versions, and most importantly debug code. Debugging lets you see inside the code, it removes the mystery and lets you take risks.

Expressive programming is not engineering, but software engineers and computer scientists are the people who design programming languages. Few languages have been developed with artists in mind. The engineering approach is to start from a well-defined problem, develop an architecture or structure for the solution, and use that to break the problem down into smaller components.

In general, artists aren’t trying to find solutions to problems; their approach is bottom up rather than top down. They start with a single, often vague, idea and then explore the possibilities that surround it, expanding and refining the original approach. This is the essence of Reas’ conception of software sketching. I’d make the observation that this is not so different from what a lot of scientists want to do with software: take some data or a mathematical concept, explore it, and see where it takes them. There are a number of popular mathematical / scientific packages built around this concept, such as *Mathematica* and *Matlab*.

In Chapter 4 (Summary 2004) I made the analogy between children’s building toys and programming languages. As a child I used to play with two building systems; *Lego* the colourful plastic building blocks that snap together, and the older metallic nuts and bolts *Meccano*. *Meccano* was developed in 1901 by Frank Hornby and is
The Materials of Software Art

representative of 19th Century iron and steel boilerplate construction techniques. Lego blocks, first developed in 1949 by Ole Christiansen, embody the 20th Century modular construction paradigm – a child of the post-war plastics industry. Lego encourages exploration and rapid prototyping of ideas, it is easy to put together, but not very strong. Meccano takes longer to put together, is strong, and if you aren’t following instructions, then you need to have a clear conceptual image of what you are trying to make. I would extend the analogy to include Meccano as representative of the engineering programming strategy, and Lego of the sketching.

Choosing A Language

At the start of 2004 I undertook a survey of potential software environments and programming languages to replace C++. At this stage my criteria were less developed than the list in the previous Expressive Programming section. I hadn’t yet articulated the idea of open-endedness and the qualities that engender it, such as richness and promiscuity. I had a greater emphasis on having an interface to OpenGL as a central requirement than I now would.

I looked at both commercial and open-source programming environments being used by artists: MAX/MSP, jMAX, PD, Director, Flash, Processing, Blender and also at some of the most popular programming languages: Perl, Python, Java. If I were doing the same analysis now, I wouldn’t have bothered with considering Director, and would have added Ruby. Other languages such as Lisp and Smalltalk, although they have powerful capabilities, seemed too marginal.224

A tabular summary of an updated analysis is contained in Appendix 3. The reality of the decision came down to informed intuition; I looked at the language specifications, sample code, and at what people were saying and doing.

---

224 Lisp dates from 1958 and is the second oldest high-level programming language, widely adopted for A.I. research. It has inspired the development of many other languages, including Smalltalk and Python. Smalltalk led to the development of Squeak, which has an application Scratch used to teach children how to program games and interactive art. In future work I plan to explore expression through these languages. For a diagram of the genealogy of the major programming languages see Leverenz, Eric, The History of Programming Languages, 2006, Available: http://oreilly.com/news/graphics/prog_lang_poster.pdf
I didn’t particularly want to use MAX/MSP, jMAX or PD, because I didn’t work effectively with the visual programming metaphor of patching they use. Director, the original multimedia-authoring environment, did have a 3D capability, but it was visually clunky, and support for its use was in decline. Flash appeared slick, but it didn’t have a built in 3D capability and at that stage no hardware acceleration, so it seemed unlikely it would have the performance to do the type of work I envisaged. Blender had a game engine that was hardware accelerated but support for it was sporadic, and the visuals from it didn’t have the quality I wanted.

Processing did look interesting; it had the advantages of Director and Flash, in that works could be shown via the Internet as Java applets. The idea of creating an open-source environment for graphic programming that allowed results to be publishable with one mouse click complete with source code, was clever. But, at that stage, it didn’t have access to 3D hardware acceleration or a 3D vector library. I concluded that none of the purpose-built software environments suited my needs, and I would have to look towards a general purpose programming language.

Java is very widely used, it is a network language, and a Java program can run on any hardware with an appropriate virtual machine. It used to have the perception of being slow, but performance enhancements in recent years have improved the speed to near that of C++. At the time that I was looking, the Java support for accelerated 3D graphics was in disarray. Java3D, a more sophisticated toolkit than OpenGL, was in decline and JOGL (OpenGL for Java) was not robust.

Perl was a scripting language that I had used extensively before, and was very compact and powerful. I would have used Perl but, at that stage, there was no interface to OpenGL. Aside from this, Perl code is easy to write but difficult to read; it uses a lot of decorators (@,#,$,%) to denote different variable types and has many ways to achieve any operation.

Python

Python emerged from this analysis as the most appealing option. Unlike Perl it had a maintained interface to OpenGL, so it met my basic requirements. Although it didn’t seem to have been much used by visual artists, it had a reputation as a scripting language for game engines and was used in the open-source 3D animation application Blender. What convinced me to seriously try it was the expressiveness of the language.
The Materials of Software Art

Expressiveness of a programming language is a measure of its ability to succinctly express a range of algorithmic concepts, generally measured in terms of the program length or size. Comparisons of Python programs to those written in other languages indicate that it is one of the most expressive languages available (6 times more expressive than C/C++ and 4 times more expressive than Java). Python has the unusual feature of being structured through indentation while, in most programming languages, code is structured through the use of punctuation such as “;” to indicate the end of a line, and brackets or braces to group statements together. Using indentation means that Python has less punctuation, and is consequently highly readable.

Guido van Rossum developed Python, with the first release in 1991. It is described as a very high level language with minimal core syntax, but with a very large standard library of modules. I have yet to come across a task for which there isn’t an appropriate Python module or library available.

In conjunction, these attributes mean that developing working applications is fast and, just as importantly, they are easy to maintain and debug. The combination of conciseness, high level of abstraction and readability means that when it is necessary to return to a program after some time, the function and structure of the program can be rapidly comprehended. In my experience of programming languages, Python has come closest to allowing the free expression of concepts.

Processing

If I were starting this project in 2008, I would undoubtedly choose to use Processing. The availability of good books, a vibrant community of artists, and the ability to easily

---


227 See sample code comparisons for calculating a Mandelbrot fractal between Python C++ and Java in Appendix 3: Comparison of sample program implementation in 3 languages.

228 With many programming languages such as C++, Fortran, Basic and Pascal, I have felt a sense of fatigue when setting out to write a program from having to type large amounts of boilerplate code. This fatigue can also interfere with debugging and program modification and refinement.
The Materials of Software Art

publish and share work online have made it a very successful open-source project. It has introduced thousands of artists and designers to programming, and sparked a resurgence of interest in algorithmic aesthetics. Reas and Fry refer to Processing as a sketchbook:

…artists working in software need environments for working through their ideas before writing final code. Processing is built to act as a software sketchbook, making it easy to explore and refine many different ideas within a short period of time.\(^\text{229}\)

Processing is built upon the Java language, but provides an environment in which the code required to construct the windowing environment and animation is hidden, and access to the Processing class library of graphic functions has been made transparent. A Processing sketch is visually less complex and easier to read than the equivalent Java program. However, Processing does have its limitations. It has the ability to utilise 3D hardware acceleration, but the functionality for working in 3D space is limited. Processing has no debugger or command line interpreter for interacting with code intimately.

When the programmer needs to go beyond these limits, they have to transition to using Processing libraries as part of a Java environment, which I think was Reas’ and Fry’s intention. Artists, as they become familiar with programming, are then able to make the transition to general programming environments. Java is cleaner and more compact than C++, but is still a language based in an engineering methodology.

The approach I have developed using a combination of Python and C++ has, for making art, several advantages over Java. It requires a greater level of technical literacy, an understanding of programming in both Python and C++, and knowing how to interface components constructed in the two languages. Analytical thinking and expression are definitely required but, aside from constructing and interfacing the core C++ components, it is possible to program using different cognitive modes. The flexibility and ease of expression is extraordinary. Programming in Python is fun; it’s a lot like having a very large box of Lego full of new and interesting bricks, all of which just snap together.

Levels of Software Abstraction

In Chapter 1 the influence of language on cognition was discussed, and it was noted that the choice of programming language influences both the types of structure a programmer constructs, but also their expression. A corollary is that the functionality of additional toolkits and modules that the programmer chooses to use has an even greater influence. This may be barely a consideration with many specialised software environments. When programming in *Actionscript* for *Flash*, for instance, there is no choice about what software you will use to draw objects on screen: the programming language is built around the display engine.

In more general programming environments, such as *C++*, there are dozens of different libraries that can be used to draw an object on screen, having capabilities ranging from simple to sophisticated. Each will also represent a different level of complexity and abstraction; a high level toolkit may take only a few lines of code to create a window and draw something in it, whereas a low level one may require dozens of lines of code to do the same thing.

Lower levels of abstraction give the programmer a greater potential range of expression and control, but require more code to be written, and may require the programmer to implement techniques that are a standard part of higher level toolkits. Conversely, a high level of abstraction in a toolkit may hide alternative modes of expression, or impose the aesthetic judgements of others.

Selecting the appropriate level of abstraction for a toolkit is a dilemma for the software artist. David Rokeby talks of the difference between programming in assembly language and using a higher-level language he developed. Using the higher-level language, he could do in hours what took him months in assembly language, but the higher-level language biases the artist toward doing things in ways that may lead to the creation of clichéd expression.

*Assembly code itself contains very little in the way of abstraction. I would take my idea and build it, as it were, atom by atom. It presents a relatively level playing field. My higher-level language was more of a terrain, with peaks and valleys. Once I was placed somewhere on that landscape, there were pathways that were easy and pathways that were difficult. My decisions about how and what to implement were inevitably*
The Materials of Software Art

influenced by this terrain. A landscape gives you a fine view in some directions and obscures others.\textsuperscript{230}

This problem comes to the heart of this project’s exploration into the conditions for expressive software. To work within a system which provides high level abstractions and capabilities biases the modes of expression. Working within systems that are too low level, requires a focus of attention and resources to achieve expression, which distracts from form and content, and can lead to empty expressions of technical virtuosity.

For many programming tasks, there is no question that the higher the level of abstraction the better. For instance, in constructing my works \textit{Communal Sense} and \textit{Stories about You}, I needed to pass data across the network. Writing a reliable client / server in C++ and sending and decoding data is a lengthy task (and irrelevant to the behaviour and visual expression of the work). With the highly abstracted \texttt{Python} module \texttt{pygnet} I used, it took only a dozen lines of code. Another task also from \textit{Stories about You} required the contents of a web site to be downloaded and specific pieces of text extracted. Again, this was a relatively simple task with \texttt{Python} modules that presented a high level of abstraction. I wouldn’t have even tried the same task in C++, because it would have been quicker to spend a day cutting and pasting the pieces of text needed from the website. And it may have been that a better selection of texts would have been found, because I would have been selecting and editing as I went.

I think that all worthwhile art requires effort and intention. Sometimes this occurs before a work is made and sometimes during the process of making. But it has to be effort applied at the right point; effort for effort’s sake just leads to fatigue.\textsuperscript{231} I have found this in nearly all the artworks I have made: there has to be a problem or task that requires time and effort, to make the process seem rewarding.


\textsuperscript{231} It so happens that in the final version of \textit{Stories about You}, I did edit each entry (but I didn’t cut and paste them together manually). It was necessary to finish the work, but wasn’t necessary in the developmental stages.
The Materials of Software Art

The solution I have come to, to avoid these abstraction-induced clichés, is to work with systems that are open-ended. In general this has meant open-source, as open-source systems encourage extension by others, as opposed to proprietary systems that are closed. Working with both Python and C++ allows for a very wide range of choice in selecting toolkits and the level of abstraction they represent.

Drawing in 3Dimensions

Selecting a software toolkit for the visual manifestation of these artworks required a similar methodology to that for selecting a programming environment. It is possible to construct a rationalised analysis after the fact, but selection required a similar intuitive mix of experimentation, developing an understanding of the terminologies, assessing imagery, applications and community. My initial thoughts were to work directly with the OpenGL toolkit, with the reasoning that this approach would allow the greatest flexibility and least bias in the visual outcomes.

OpenGL

The OpenGL graphics library has been under development for a long time. It was originally developed by Silicon Graphics, the company renowned for its pioneering work with animation and virtual reality tools in the 1980s and early 1990s. The only comparable software is Microsoft Direct3D, which is only available on the Microsoft Windows platform. Higher-level game engines and graphic toolkits are built upon one or both of these libraries.

OpenGL is a low level library; it implements a particular way of rendering imagery and constructing three-dimensional space. It provides a common abstraction for interacting with computer graphics hardware, and is available on almost all computer platforms from mobile phones to supercomputers. It provides the means for representing geometric objects in two and three-dimensional space, and for transforming image data onto the surface of those objects. It provides a basic model of lighting and shading and, in recent years, extensions to the OpenGL standard have incorporated the usage of geometry, vertex and pixel shaders - small programs that can modify the way that the graphics engine renders form, surface and light. These can be used to produce more sophisticated visual experiences such as shadows, better lighting, and complex surfaces. In my naive early encounters with OpenGL, it took
me some time to realise that there were many things that it could not do inherently, such as rendering correct transparency and shadows.

**Scene Graphs**

I was finding *OpenGL* increasingly awkward and cumbersome. For instance, in *Stories about You*, if I wanted to render correct transparency I would have to develop an algorithm that broke my ribbons of text into separate polygons, sorted them by their distance from the camera and then drew them in depth order. This was not impossible or even difficult, but it would have been time consuming.

I spent some time looking at the possibility of using a game engine (there are a large number of open-source game engines available) that would provide a higher level of graphic abstraction. However, I wasn’t designing a game, and game engines are generally developed with a particular style and type of game in mind, which led me to reject this approach as too limiting.

I found out that what I was looking for is referred to as a *scene graph*, a data structure that organises and displays a hierarchical collection of geometric objects. Scene graphs are found in drawing programs like *Adobe Illustrator*, but are also used in most modern games. The ancestor to the type of scene graph I wanted to use was developed by computer graphic pioneer *Silicon Graphics* in the 1990s, and came in two forms, *Inventor* (ease of use) and *Performer* (high performance), each developed to do different tasks. I eventually identified five open-source cross platform scene graphs which used hardware graphic acceleration: *Coin 3D*, *OpenSG*, *OpenSceneGraph*, *Ogre3D* and the 3D modelling and animation package *Blender*. A tabular comparison of these packages is in Appendix 3.

After trying most of these packages I chose *OpenSceneGraph*, and it has since been endorsed by the *OpenGL* foundation as the open-source scene graph that they recommend. It had an ugly web site and virtually no documentation for the software, so I passed it by several times, but kept finding myself being directed back to it. It did have a wealth of examples which demonstrated each of its features, and it also had capabilities that none of the other candidates had, such as being able to tile a display across several monitors.
The Materials of Software Art

The characteristics of OpenSceneGraph that make it suitable for a wide range of visualisation tasks are:

- Ability to scale from a single or multiple windows to complex multi-screen displays.
- Efficient high performance implementation.
- Active development and user community. Development keeping pace with innovation and development of the underlying OpenGL library.
- Applicable to 2D and 3D graphic applications.
- Scene graph design is not biased towards a particular kind of geometric visualisation.
- Able to load and output a variety of image types, fonts and object formats.

The Materiality of Numbers

Scripts and Scores and Experiential Flow

A script in digital media is the expression of an algorithm in a formal language. Scripting, as opposed to programming, usually takes place in a rich environment or framework (such as the Adobe Flash multimedia package) of existing capabilities and entities (frames, events, objects) and allows for the rapid expression of an idea. It is a process more akin to sketching than engineering; that is, the process of development determines the outcome, rather than an initial specification.

A reinterpretation of the algorithm developed within one scripting environment, will be similar, but different when translated into another. Like the interpretation of a theatrical script by different actors the words may be the same, but the nuance and intent change the expression of an idea.

A musical score describes the duration and pitch of a sequence of notes. In a complete score it will also describe the speed of execution, the type of instrument, and even when the performer should breathe. It is essentially an algorithm for the construction of a piece of music. However, there is a vast difference between

---

232 A formal language has an explicitly fixed syntax and limited verbs.
The Materials of Software Art

mechanical interpretations of a score (via a sequencer-synthesiser), and a musical performance.

Software art... embodies the essential form-making capabilities of the artist, somewhat as a musical score embodies the musical ideas of a composer.\textsuperscript{233}

The score is a symbolic representation; it is generally possible to reconstruct the score from the performance, but not the performance from the score. The symbolic content of a score (or text) has an information flow approximately ten million times less than the real life experience of a performance.\textsuperscript{234}

In software art, the program is the score, but there can only ever be a mechanical interpretation of that score; when the artwork is made manifest by the execution of the program, unless the artwork employs some element of performance or interactivity.

However, reading a book (a sophisticated score) is often a more rewarding experience than watching a film adapted from that book. Scores can have expression too.

\section*{Sampling and Synthesis}

Two distinct strategies have emerged for representing and constructing representations in digital media: \textit{sampling} and \textit{synthesis}.

Sampling is the process whereby a physical medium (for instance sound or image) is reduced to numbers, a symbolic representation.\textsuperscript{235} Sampling can occur in either or

\begin{table}
\centering
\begin{tabular}{|c|c|}
\hline
\textbf{Format} & \textbf{Data rate \textasciitilde}MB/s \\
\hline
Text & 0.00015 \\
Audio (single channel) & 0.044 \\
Audio Surround & 0.528 \\
Video & 30 \\
Cinema & 250 \\
Real Life & 5000 \\
\hline
\end{tabular}
\end{table}

\textsuperscript{233} Verostko, "Epigenetic Painting: Software as Genotype, a New Dimension of Art". p18.
\textsuperscript{234} Consider an oration and its reception in a variety of forms:

\textsuperscript{235} No sampled \textit{copy} is identical to the original – only some properties are sampled. For example, a desktop scanner samples the way an image original reflects light at particular wavelengths, but not variations in surface depth. The sampled image is a representation, but how complete is it?
The Materials of Software Art

both of the space and time domains. For instance, a sound may be sampled at a
certain depth (numerically the number of possible sample states) in the time domain
(with samples being taken a number of times per second). Sampling requires a
mechanical instrument, a transducer, to do the sampling. The quality or faithfulness
of the sample depends not just upon the density of samples, but the also upon the
transducer. The sampled media can then be reconstituted via another transducer, for
example, on screen as a visual representation of an audio waveform, via speakers as a
sound. An assumption is often made that, if the sample is dense enough and has
enough depth, then, for all intents and purposes it is indistinguishable from the
original, rather than being a symbolic representation of particular qualities of the
original.

Synthesis is an algorithmic approach to generating media. For instance, the Perlin
noise algorithm is widely used to generate convincing simulacra of texture ranging
from clouds to wood textures.\textsuperscript{236}

Both approaches have their advantages and disadvantages. Sampled media can be
inflexible and even brittle if not handled carefully, whereas synthetic media, whilst
flexible, often has a very distinctive signature.\textsuperscript{237} For example, fractals offer an infinite
variety of shape and form, but are always recognisable as fractals. Many digital media
applications now use combinations of synthetic and sampling techniques, such as
those for 3D animation, which use sampled images to create realistic surface textures,
while using algorithms to synthesize lighting and atmospheric effects.

The Quality of Randomness

There is a lot of conflicting discussion about randomness and random number
functions in generative art practice, some of which has a similar flavour to discussion
about determinism in interpretations of quantum mechanics. The concept of
randomness and its relationships to entropy (order and energy) and complexity is a
subtle mathematical quality with both practical and philosophical ramifications.

\textsuperscript{236} Ken Perlin, \textit{Noise and Turbulence}, 2002, Available:
http://www.mrl.nyu.edu/~perlin/doc/oscar.html

\textsuperscript{237} Brittle in the sense that if some quality of the media is stretched too far, then it will exhibit its
sampled origins through pixilation or banding.
The Materials of Software Art

Burroughs, Ginsburg and Cage embraced chance as a generator for the aesthetic process. However, software random number generators are deterministic, that is, for a given starting point or seed, a particular random number generator will produce identical sequences of numbers. The common practice is to use a time function (generally seconds since the epoch) to generate a seed for the random number generator so that the execution of a program will produce unique results each time, preserving the appearance of true randomness. John Maeda famously rails against the use of random number generators in software art and design; he sees randomness as a lazy way of trying to introduce complexity and freshness, but declares that it introduces no new information into a system. His former student, Golan Levin, qualifies this stance, and places randomness as an element to be used below the level of perception, for instance in the use of textures.238 Artist David Rokeby takes a more philosophical stance:

\textit{Opposed to the incredibly compressible complexity of fractals is the complexity of true randomness. Something can be said to be random if it cannot be expressed by anything less than itself... that is to say, it's incompressible.}\textsuperscript{239}

He then goes on to discuss randomness in the computer:

\textit{Randomness and noise are usually things we avoid, but in the purely logical space of the computer, randomness and noise have proven to be welcome and necessary to break the deadly predictability. But random number generators, used so often to add human spice to computer games and computer-generated graphics are not random at all. They merely repeat over a fairly long period; a sterile simulation of the real thing.}\textsuperscript{240}

Rokeby here is highlighting the difference between physical randomness, and mathematical pseudo-random functions; both may have similar statistical qualities, but they are not the same thing, pseudo-random functions are cyclic, they repeat, often over very large, but still finite numbers. He goes on to discuss how the chance occurrence of the works of Shakespeare through random events (the many monkeys on many type writers), is what has occurred though the action of sub-atomic particles, molecules, life-forms, and societies over time.


\textsuperscript{239} Rokeby, "The Construction of Experience: Interface as Content". p42.

\textsuperscript{240} Rokeby, "The Construction of Experience: Interface as Content". p42.
On the other hand, neither a fractal nor a pseudo-random number generator is capable of this feat. Those systems are closed. No matter how far you expand them, Shakespeare’s work will not be generated.\textsuperscript{241}

Rokeby is implying here that there is a different quality to the randomness inherent in the physical world, to that generated by a pseudo random number generator. His argument comes down to his conjecture that digital randomness is a limited approximation of the ‘power of irresolvable, not-fractal complexity.’\textsuperscript{242}

This argument can be further explored through discussion of the A-life system \textit{Tierra}.\textsuperscript{243} \textit{Tierra} was developed as a simulation of an ecosystem by ecologist Thomas S. Ray in the early 1990s in which computer programs evolved through competing for processing time and memory resources. The system was hailed as an evolutionary system, and it was conjectured that it could breed increasingly complex and even intelligent life forms. However, analysis has since shown that the system has only a very limited capacity for developing complexity: ‘Algorithms can never create information, only destroy it.’\textsuperscript{244}

Maeda’s insight might be better expressed as: it is a mistake to confuse pseudo-randomness with novelty. Novelty in works of art and design implies not just difference, but difference in kind. A similar analogy could be made between interactivity and engagement in new media. During the first flush of enthusiasm for interactivity in the early 1990s, there seemed to be a widespread belief that just adding interactive elements would make a work more engaging.

My own view is closer to Golan Levin’s: pseudo-randomness should be recognised as a material - a quality - not as a thing. It has characteristics dependent upon the function that generated it. Random number generators can be described by their probability distribution and spectral profiles. For instance, two functions may generate two-dimensional visual noise with the same average value and similar tonal distributions, but have different spectral characteristics. A blue (high frequency)
noise dithering screen applied to an image will have greater clarity than one that has redder (low frequency) spectral components.

Pseudo-randomness is a material. To apply it without consideration for its properties leads to a greater uniformity of behaviour or appearance; it is not a generator of novelty or complexity but can, with consideration, add texture and support the semantic layering of the work. Randomness can be manipulated through the following:

- Choice of distribution profile. The standard pseudo-random noise function is chosen to have a statistically flat distribution, an equal chance of all numbers occurring. This is desirable for Monte Carlo style simulations, (although we may prefer the function to have a blue rather than a red spectrum). However, if the simulation were to create a population of characteristics (say height), then our randomness should have a Gaussian bell-shaped distribution.
- The pseudo-random number distribution can be manipulated through further mathematical or algorithmic computation. For instance, simple functions such as an inversion, square or square root will modify the shape of a pseudo random number generator to approximate other distributions. A pseudo-random number generator might be used to generate a two-dimensional probability density by generating random numbers and comparing them to thresholds obtained from an image.

The works that make up this thesis have all used pseudo-random numbers in one of the following ways:

- In generating the positions for elements to be drawn to render an image. These random positions may be accepted or rejected based on other criteria that the drawing algorithm imposes.

---

245 Monte Carlo methods use random numbers to numerically calculate approximations to problems that are difficult to solve analytically. For example, calculating the area of a complex shape.
The Materials of Software Art

• In determining the sequences of events. Rather than an animation playing through a pre-determined sequence of key images, the sequence order is determined randomly.

• To make logic fuzzier. Random numbers can be tested against a threshold to simulate interactions based on probabilities.

Random numbers can reduce repetition, but they can’t generate novelty.
Chapter 6 - Conclusion

This exegesis began by considering David Rokeby’s description of the computer as a ‘wishful self-portrait’; a machine that embodies a collection of abilities that we aspire to, but do not possess in large measure. I would extend his metaphor to argue that the significance of the computer is that it has provided a mirror in which our ideas about the nature of consciousness, cognition and language can be examined.

Drawing upon diverse sources from neurology, psychology, linguistics, philosophy and cognitive science, a new understanding of cognition is emerging and, while it is too early to call it a coherent theory, the sources from which this new understanding is drawn share common threads:

- The mind is embodied and not an artefact of a symbol-processing engine.
- Human language and communication are dependent on personal context rather than on an inherent code.
- Cognition can extend beyond the organ of the brain to include our bodies and artefacts.
- Underlying our learned symbolic associations, we think with sensate images, not internal symbolic representations and, in this sense, visual art can be a direct representation of thought.

From this perspective, it is hard not to make analogies to events one hundred years ago when the certainties of knowledge of the physical world were shaken by the concepts of relativity and quantum mechanics. It is no coincidence that the way we pictured the world began to change at the same time, or that those changing visual representations helped formulate new physical understandings.246

The common denominator between art and science is the degree to which expressions in each domain are compatible with the human mind. Einstein’s elegant theories of the universe are true (and beautiful) because they are consistent with the capability of the human mind to understand such ideas. Picasso’s application of new techniques, juxtapositions, and perspective are beautiful (and true) because they see the mind: they are consistent with the capacity of the human mind to understanding these visual


133
Conclusion

As scientists discover laws of the universe that are congruent with mind, artists discover visual images of the world that are harmonious with mind. Both explore the truth and beauty of the mind and, at an abstract cognitive level are identical.247

Software art is sited at the intersection of sensate expression and formal logic – two diametrically opposed modes of thought, one requiring a lateral integration of mental processes, the other a disciplined abstract focus. The difficulties that conjunction presents are why software art has a role to play in the development of the new conception of mind, cognition and communication. If visual art is the direct expression of sensate images, then software art is uniquely placed to examine the relationships between sensate imagery and symbolic expression. For instance, while software art cannot confirm or deny a mathematical description of complexity it can, through artworks, test to see whether that description of complexity is congruent with our perception of meaning. It does this through the visual representations of new forms such as fractals, emergence, interactivity and a-life, and is where the significance of these concepts can be evaluated and placed within a broader understanding.

This project has used these emerging views of language and cognition to examine the conditions and materials of software as a medium for the expression of sensate imagery. The works that have been developed to form the thesis are not illustrations of particular linguistic or cognitive theories, but are the artefacts of the sensate thought processes I have undertaken in developing these representations of language and thought. In this exegesis I have documented how the development and resolution of each work has entailed a journey to a greater understanding, which has in turn directed my reading and evaluation of theoretical positions.

The research has been carried out through practical experimentation, and this is where the beauty (and truth) of these ideas can be tested. Cognitive embodiment frees the mind from the cages of absolutism and relativism; truth and beauty are human intuitions. They are certainly dependent on personal context but, like most language concepts, there is a surprising degree of consensus about what concepts and artefacts evoke the sensations.

Conclusion

An attribute our ‘wishful self-portrait’ currently lacks is the ability to analogise. Training and practice have enabled us to approximate thinking and expression with symbols so well that we have confused representation with the referent. This is what the computer reminds us of, if we examine it closely: code is not language; language is not a collection of codes. I have argued that language is an inappropriate metaphor for the symbolic constructions of programming, that children’s building systems with their limited syntax and connecting together of pre-made objects is more appropriate.

Through the development of the artworks that comprise this thesis, and which promote a methodology for expression through software, a number of insights about software have been documented. Criteria for an expressive programming methodology have been articulated; these include:

- The quality of open-endedness, which includes the ability to extend the environment by integrating external code objects. Open-endedness means not being limited to using one toolkit.
- Expressiveness of the programming language; a combination of the clarity and compactness of the written code.
- The ability to work with systems and toolkits on a number of levels (from low level to high level abstraction), and the experience to recognise what is the appropriate level for an expressive intention.

Consideration has been given to the material qualities of software, particularly the relationship of randomness and emergence to their algorithmic analogues. A conclusion from this examination is that algorithms are not generators of novelty; they are a material or support on which the semantic associations of the work rests. It is only with input from the physical world that the potential for novelty exists (for example, the Internet is only interesting because of the billion who use it and contribute to it). Software art, along with a-life and artificial intelligence confront many interesting questions about the nature of computation, such as: What are the

---

248 I conjecture that computational emergent behaviour has a similar relationship as randomness between physical and computational realms. Like pseudo-random number functions, computational emergence should be thought of as a kind of pseudo-emergence, indistinguishable for most applications, but with a different level of fundamental complexity.
Conclusion

limits of simulation? What does true randomness imply? How does creativity and novelty emerge?

This project set out to explore the display technologies that consist of the combination of powerful dedicated 3D graphics processors and flat panel displays, with the intuition that these will have a significant place in visual culture in the 21st century. Over the course of this project there have been five generations of graphic accelerators released, and the cost of LCD displays has fallen by a factor of more than ten. These technologies enable a seamless merging of the strategies of sampling and synthesis, and will engender an increasing complexity of real-time visual imagery for the foreseeable future. Underlying these technologies are a set of assumptions about the representation of space and the modelling of objects by light, which are not easily subverted.

No program is ever completely finished, and this is also true for the artworks developed for this thesis. They have reached a stage of visual and conceptual resolution, however, to advance them further they will require the deep reflection that has occurred through reviewing this project. As with most research activities this research project has identified as many avenues for further investigation as it has explored.

I concur with Reas and Fry when they call for computer literacy to mean not just reading (being able to use software that others have made) but also writing software. To have a successful relationship with this cognitive artefact, the computer, we must be able to express our thoughts through it. For artists that includes the ability to subvert and direct the processes of the machine, and this usually involves programming at some level. I believe, and the outcomes of this research demonstrate, that the expressiveness and promiscuity of the programming language used remove many of the barriers to free expression in this medium.

For me, the essential question for software art is this – can we use knowledge to make pictures with computers that illuminate the beauty (and truth) of being? And in doing this come to know what it is to be human, in a new way.

Appendix 1) Software Used in Artworks and Website

The following table lists the key software used in this project. As programs developed, different modules were used for the same function, e.g. NumPy replaced NumArray. The modules listed are the ones used in the completed works. General software packages for editing images and word processing are not listed.

<table>
<thead>
<tr>
<th>Name</th>
<th>URL</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python</td>
<td><a href="http://www.python.org">http://www.python.org</a></td>
<td>Very High Level Language – Includes extensive library of modules</td>
</tr>
<tr>
<td>Eric IDE</td>
<td><a href="http://www.die-offenbachs.de/eric/index.html">http://www.die-offenbachs.de/eric/index.html</a></td>
<td>Integrated Development Environment for Python (Editor / Debugger)</td>
</tr>
<tr>
<td>OpenSceneGraph</td>
<td><a href="http://www.openscenegraph.org">http://www.openscenegraph.org</a></td>
<td>3D Scene Graph and Windowing toolkit</td>
</tr>
<tr>
<td>NumPy</td>
<td><a href="http://numpy.scipy.org">http://numpy.scipy.org</a></td>
<td>Matrix additions to Python</td>
</tr>
<tr>
<td>OpenCV</td>
<td><a href="http://opencvlibrary.sourceforge.net">http://opencvlibrary.sourceforge.net</a></td>
<td>Open Computer Vision Library</td>
</tr>
<tr>
<td>SWIG</td>
<td><a href="http://www.swig.org">http://www.swig.org</a></td>
<td>Wrapping generating software for binding to foreign objects</td>
</tr>
<tr>
<td>PIL</td>
<td><a href="http://www.pythonware.com/products/pil">http://www.pythonware.com/products/pil</a></td>
<td>Python Imaging Library</td>
</tr>
<tr>
<td>pyrund</td>
<td>Not publicly released</td>
<td>Python wrapping for OpenSceneGraph</td>
</tr>
<tr>
<td>C/C++</td>
<td><a href="http://www.gnu.org/software/gcc/">http://www.gnu.org/software/gcc/</a></td>
<td>GNU C and C++ compilers</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>URL</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>SciPy</td>
<td><a href="http://www.scipy.org">http://www.scipy.org</a></td>
<td>Scientific Python libraries</td>
</tr>
<tr>
<td>pygnet</td>
<td><a href="http://code.google.com/p/pygnet">http://code.google.com/p/pygnet</a></td>
<td>Simple library for game networking (built on top of Twisted networking library)</td>
</tr>
<tr>
<td>pYsearch</td>
<td><a href="http://pysearch.sourceforge.net">http://pysearch.sourceforge.net</a></td>
<td>Module to access yahoo search engine</td>
</tr>
<tr>
<td>Shm module</td>
<td><a href="http://nikitathespider.com/python/shm">http://nikitathespider.com/python/shm</a></td>
<td>Shared Memory module</td>
</tr>
<tr>
<td>Gnosis Utilities</td>
<td><a href="http://gnosis.cx/download/gnosis">http://gnosis.cx/download/gnosis</a></td>
<td>XML utilities module</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>URL</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drupal CMS</td>
<td><a href="http://www.drupal.org">http://www.drupal.org</a></td>
<td>Drupal Content Management System for Web site</td>
</tr>
<tr>
<td>Adobe Flex</td>
<td><a href="http://www.adobe.com/products/flex">http://www.adobe.com/products/flex</a></td>
<td>Programming environment for Flash</td>
</tr>
<tr>
<td>Adobe Flash</td>
<td><a href="http://www.adobe.com/products/flash">http://www.adobe.com/products/flash</a></td>
<td>Animation environment for Flash</td>
</tr>
</tbody>
</table>
Appendix 2) Algorithms

Spring Simulation

Spring simulation as a method to attract an object (such as a piece of text) to a target point. Using Hooke’s law of elasticity for a linear spring gives:

\( F = -kx \)

So if we assume that the spring goes between two points \( p \) and \( q \) where \( p \) is the particle moving at velocity \( v \) and \( q \) the fixed target and the spring has a rest length of 0, and that \( k_s \) = the spring constant and \( k_d \) = the spring damping constant and where \( \Delta x \) = the distance between \( p \) and \( q \) as a vector, then the force on the particle \( p \) is:

\[ F_p = -k_s \Delta x - k_d \cdot v \]

and the acceleration of the particle from Newton’s third law, for a particle \( p \), with mass \( m_p \):

\[ a = F_p / m_p \]

Then using a Verlet integration scheme,\(^{250}\)

\[ p_{i+1} = 2p_i - p_{i-1} + a \cdot \Delta t^2 \]

(where \( \Delta t \) is the change in time between \( p_{i+1} \) and \( p_i \)) the movement of the particles can be easily calculated.

Empirical Observations

The quality of movement is controlled by the spring constants. The spring strength \( k_s \) controls the speed and size of the oscillation, while the damping constant \( k_d \) controls the degree of bounce and the time taken to converge on the target.

The fastest rate of convergence to the target occurs when the ratio between damping and spring strength is around 1.2

I’ve also tried using functions for modifying spring strength or damping with time, such as gradually ramping up the spring strength, so that when a new target is selected the old image doesn’t fly apart explosively; or if it is bouncing too much and not settling down, then the damping increases after a certain time.

\(^{250}\) Jakobsen. "Advanced Character Physics".
Algorithms

In *Ocular Coincidences* I initially tried a single spring and damping constant, and then tried random values for all particles. Neither was satisfactory: one led to the particles pulsing in step, the other to a random cloud. In the end, I tried assigning just four or five different values so that groups of the particles pulsed together giving some cohesion without being too random. To get the words to twist, rather than simulating a model of angular momentum as I did in the original *Memetic Variations*, two springs are used for each animated word, with two target points calculated from the angle that the word is supposed to be placed at. This means that a vector can be calculated from the two springs to determine an angle at each time step. Additionally, the distance between the two spring anchors could be used to calculate the size of the text. If multiple springs were used this way (say one for each letter), with slightly different spring constants, then wobbly writhing text could be created as I've done in *Stories about You* to place the text onto contours.
Brownian Motion Simulation

Brownian motion is the random movement of particles suspended in a liquid. It is a stochastic process that can be modelled simply using statistical functions.

Calculate the initial x and y positions for the particles using a uniform random function.

Calculate initial velocities for the particles using a normal (Gaussian) distribution.

Calculate a time length for each particle using an exponential distribution.

At each time step:

Move the particles.

Decrement the time by the time step for all particles.

For those particles whose timescale is zero or less:
    Calculate a new velocity using a normal distribution.
    Calculate a new time length using an exponential distribution.
Algorithms

**Drawing Algorithm #1**

Draw a print using tens of thousands to hundreds of thousands relatively small words aligned by the tone and gradient of a source image.

Construct a function which maps and interpolates the dimensions and tone of a small image (say 250 x 400) pixels to a large image (12500 x 20000) pixels, for a given set of coordinates in the large image plane - it will return a tonal value corresponding to the small image.

Construct another function that calculates the angle of the gradient for a given set of coordinates in the large image plane using the tonal function.

Fill the large image with a neutral tone.

Select a text.

For each word in that text (repeat text until image is dense):
- Pick a random point on the image surface.
- Use the first function to calculate a tonal value for the text.
- Use the second function to align the text.
- Composite the text image with the image.
Drawing Algorithm #2

First calculate key positions to form an image by aligning words by tone and gradient of a source and making darker words larger. Then animate words along cubic splines through the key positions calculated from a sequence of images.

Construct a function which maps and interpolates the dimensions and tone of a small image (say 100 x 80) pixels to a larger image (1024 x 768) pixels; for a given set of coordinates in the large image plane it will return a tonal value corresponding to the small image.

Construct another function that calculates the angle of the gradient for a given set of coordinates in the large image plane, using the tonal function.

Select several hundred words from a dictionary.
Select a dozen small simple greyscale images.
For each image calculate the positions of the text by:

For each word:

Pick a random point on the image surface.
Use the first function to calculate a tonal value for the text.
If the tonal value is dark make the text larger, if it is light make it smaller.
Use the second function to align the text.
Store the position size and angles for the text.

Construct a function that calculates a cubic spline between each of the positions, scales and angles for the values calculated for each image.

Define a function that relates frame number to a position along the spline (how fast the animation runs, and how many frames in the animation).

For each frame:

For each word:

Use the spline function to calculate the position, size and angle of the word and add it to the frame image.

Save the frame image.
Drawing Algorithm #3

Calculate key positions to form an image by aligning words by tone and gradient of a source image.

Construct a function which maps and interpolates the dimensions and tone of a small image (say 100 x 80) pixels to a larger image (1920 x 1200) pixels; for a given set of coordinates in the large image plane it will return a tonal value corresponding to the small image.

Construct another function that calculates the angle of the gradient for a given set of coordinates in the large image plane, using the tonal function.

For each animation path:

Pick a random point on the image surface.

Use the first function to calculate a tonal value for the point.

Use the second function to align the point.

Store the key.

Drawing Algorithm #4

Calculate key positions to form an image by aligning words by the tone and gradient of a source image and setting the size of the word to be larger where darker.

Construct a function which maps and interpolates the dimensions and tone of a small image (say 100 x 80) pixels to a larger image (1920 x 1200) pixels; for a given set of coordinates in the large image plane it will return a tonal value corresponding to the small image.

Construct another function that calculates the angle of the gradient for a given set of coordinates in the large image plane, using the tonal function.

For each animation path:

Pick a random point on the image surface

Use the first function to calculate a tonal value for the point.

Use the second function to align the point.

Set the scale to a random value proportional to the blackness of the image (white is very small).

Store the key.
Algorithms

Drawing Algorithm #5

Draw a print with words on a neutral background by first drawing the background, using words aligned to the gradient and tone of a source image and progressively reducing their size, and then drawing edges using small dark text.

Create a canvas and fill with neutral grey.
Load in a small reference image to draw, and calculate a scale factor to the canvas.
Find edge vectors through a difference of Gaussians.
Interpolate the reference image to fit the canvas.
Find the Gaussian gradient of the interpolated image.
Find corner pixels in the image.
Set a maximum word size.
For each word in list (less the number of words required to draw the edges):
    Select a random point on the canvas.
    Find the tone at that point.
    Find the image gradient and align the text to it.
    Scale the word to be inversely proportional to the square root of the number of words drawn.
    Scale the transparency of the word to be proportional to the previous scale.
    Draw the word.
Set the drawing tone to dark gray and slightly transparent.
For each edge:
    Scale the edge coordinates to the canvas.
    Calculate the length of the edge, and find a word of appropriate length.
    Align the word to the edge and draw.
Algorithms

Drawing Algorithm #6

Calculate key positions to form an image by aligning words along arcs by finding simple contours along a source image gradient and setting the size of the word to be proportional to the length of the arc.

Fill canvas with a background colour.
Let N be the number of words to be plotted.
Calculate a spline interpolation function to scale the image to the canvas.
Calculate linear transformation to locate the scaled image in centre of the canvas and within the bounds of the canvas.
Find N random sets of co-ordinates that are different tonally from the background colour.
For each coordinate calculate a contour by:

Starting from a minimum radius:

Search around a circle of that radius based on the selected co-ordinate, and find where the tone of the co-ordinate intersects the circle.

Calculate the angle the intersections make to the centre.

If the angle is not too acute, and the radius is less than the maximum radius, increase the radius and repeat the previous two steps.

If the angle of the minimum radius is too acute, make it straight and align the word to the tonal gradient of the image.

Store the resulting two end points and centre as a key position, and calculate the length of the arc numerically.

Assign long words to long arcs and short words to short arcs.
Assume that long arcs are flatter areas of tone as the text will be larger and make them less opaque.

Store the keys.
**Drawing Algorithm #7**

Calculate key positions to form an image by finding random positions in a source image that differ from the background tone by a random amount.

Construct a function which maps and interpolates the dimensions and tone of a small image (say 100 x 80) pixels to a larger image (1920 x 1200) pixels; for a given set of coordinates in the large image plane it will return a tonal value corresponding to the small image.

Let N initially be a list of the number of animation paths.

Repeat the following until the number of elements in N is 0:

- Randomly generate coordinates for each element in N.
- Randomly generate a threshold for each co-ordinate.
- Let N be the coordinates where the threshold is less than the tonal value from the source image.

Store the list of coordinates as a key.
### Appendix 3) Comparison of Different Programming Environments and Toolkits

#### 1) Comparison of Characteristics for Expressive Programming Environments.

<table>
<thead>
<tr>
<th>Language / environment</th>
<th>C++</th>
<th>Java</th>
<th>Processing</th>
<th>Macromedia (now Adobe) Flash</th>
<th>MAX/MSP / JMax / PD</th>
<th>Macromedia (now Adobe) Director</th>
<th>Perl</th>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interactive shell or console</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Not interactively</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Introspection (Determine methods and usage at runtime)</td>
<td>Not by default</td>
<td>Yes, but not usage</td>
<td>Yes, but not usage</td>
<td>Yes, but not interactively</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes, exemplary</td>
</tr>
<tr>
<td>Expressiveness / Clarity</td>
<td>Verbose, and visually cluttered with punctuation</td>
<td>Simpler than C++, but still verbose and overly punctuated</td>
<td>Built on Java but class access for animation and drawing are simpler. Punctuation is the same</td>
<td>Better than Java, but mixed visual / scripting paradigm makes finding scripts confusing</td>
<td>Visual paradigm only works for less complex examples</td>
<td>Lingo has been through several generations of programming style, currently similar to flash</td>
<td>Can be if you use restraint, confusing range of ways of referring to data types.</td>
<td>Yes, very clear and concise, unusual structuring via indenting, minimal punctuation</td>
</tr>
<tr>
<td>Housekeeping (strong typing and garbage collection)</td>
<td>No</td>
<td>Garbage Collection, but requires variables to be declared</td>
<td>Garbage Collection, but requires variables to be declared</td>
<td>Garbage Collection, but requires variables to be declared</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
## Comparison of Different Programming Environments and Toolkits

<table>
<thead>
<tr>
<th>Language / environment</th>
<th>C++</th>
<th>Java</th>
<th>Processing</th>
<th>Macromedia (now Adobe) Flash</th>
<th>MAX/MSP / JMax / PD</th>
<th>Macromedia (now Adobe) Director</th>
<th>Perl</th>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>Richness</td>
<td>There are many libraries available for C++, but the standard library is not extensive. Adding functionality requires work.</td>
<td>Yes, lots of add on tools and libraries.</td>
<td>Enthusiastic user community, with a growing number of add on modules, but beyond these requires reverting to Java</td>
<td>Large user community, but many add-ons are commercial/</td>
<td>Enthusiastic arts based user community</td>
<td>Support for Director has been waning for years, old extensions no longer work.</td>
<td>Yes, large library of add on modules and projects</td>
<td>Yes, very large build in libraries of capabilities through modules.</td>
</tr>
<tr>
<td>Cross platform</td>
<td>Yes</td>
<td>Yes, wide variety of java toolkits; not as easy to link to code libraries</td>
<td>Yes, same as for Java</td>
<td>Linux support poor</td>
<td>Yes, but open-source versions seem cantankerous</td>
<td>No Linux support</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Community</td>
<td>Very widely used</td>
<td>Java is currently the most popular programming environment</td>
<td>Yes, enthusiastic community of several thousand artists / designers</td>
<td>Yes, community of developers / designers; more commercially orientated, less open-source code available.</td>
<td>Enthusiastic user community of musicians / artists</td>
<td>Dwindling</td>
<td>Large user community; Perl was the dominant scripting tool for operating systems. Large archive of modules CPAN</td>
<td>Enthusiastic user community; simple automated module installation</td>
</tr>
<tr>
<td>Interface other languages</td>
<td>Yes</td>
<td>Yes, but web delivery becomes more difficult.</td>
<td>No, but libraries have been built for communicating to hardware and other software commonly used by Artists, such as PD</td>
<td>No</td>
<td>Yes (PD especially)</td>
<td>Not easily</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
# Comparison of Different Programming Environments and Toolkits

<table>
<thead>
<tr>
<th>Language / environment</th>
<th>C++</th>
<th>Java</th>
<th>Processing</th>
<th>Macromedia (now Adobe) Flash</th>
<th>MAX/MSP / JMax / PD</th>
<th>Macromedia (now Adobe) Director</th>
<th>Perl</th>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDE</td>
<td>Many - XCode, KDevelop etc</td>
<td>Yes - several, Eclipse / NetBeans</td>
<td>Yes (but no debugger built in)</td>
<td>Yes, but confusing at the time. No debugger at the time.</td>
<td>No debugger I could find.</td>
<td>Yes</td>
<td>Yes, built in debugger</td>
<td>Has built in debugger, and a number of IDEs</td>
</tr>
<tr>
<td>Speed / Performance</td>
<td>High</td>
<td>Middle (there are benchmarks that suggest, with some implementations, Java approaches the performance of C++)</td>
<td>Middle</td>
<td>Slower than Java</td>
<td>These applications are made for real time response, but processing performance is difficult to gauge.</td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>Hardware Accelerated Graphics</td>
<td>Yes</td>
<td>Yes, several scene graphs available</td>
<td>Yes, built in, but no Scene-graph functionality</td>
<td>High quality graphics, but not 3D; now being developed to use hardware acceleration</td>
<td>Limited access to OpenGL, through GEM / Jitter</td>
<td>3D through Director 3D; I've never been convinced though.</td>
<td>No active project at the time for OpenGL access.</td>
<td>Yes, active OpenGL library project</td>
</tr>
<tr>
<td>Comments</td>
<td>Regarded as a mid level language</td>
<td>The strength of Java is its cross platform code delivery, but to maintain that strength requires using only Java.</td>
<td>Fantastic tool for sharing dynamic visual ideas via the Internet. Sister projects of Arduino and Wiring provide a way to interact with sensors.</td>
<td>Flash/Flex has improved tremendously as a programming environment</td>
<td>I don’t think the MAX way.</td>
<td>The original multimedia environment, which has been dying a slow death</td>
<td>Perl is famous for rapidly producing difficult to maintain code, because there are many ways to express the same idea.</td>
<td>The IDE situation was not as good as I first thought but now, with some effort, is reasonable.</td>
</tr>
</tbody>
</table>
2) Summary of 3D software packages evaluated

<table>
<thead>
<tr>
<th>Package</th>
<th>Location</th>
<th>History</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coin3D</td>
<td><a href="http://www.coin3d.org">http://www.coin3d.org</a></td>
<td>Compatible with Open Inventor</td>
<td>Developed by Company SIM, which release a free version. Focus is on ease of use rather than performance.</td>
</tr>
<tr>
<td>OpenSG</td>
<td><a href="http://opensg.vrsource.org">http://opensg.vrsource.org</a></td>
<td>Developed from Microsoft Fahrenheit project (next generation from Inventor and Performer)</td>
<td>Technical description sounds impressive, but I was never able to get it to work properly.</td>
</tr>
<tr>
<td>OpenSceneGraph</td>
<td><a href="http://www.openscenegraph.org">http://www.openscenegraph.org</a></td>
<td>Developed from SGI Performer</td>
<td>Used widely in the US for flight simulators and other visualisation projects. Documentation was poor, but with a great set of example programs and active community.</td>
</tr>
<tr>
<td>Ogre3D</td>
<td><a href="http://www.ogre3d.org">http://www.ogre3d.org</a></td>
<td>Not described as a scene graph, but as a general 3D graphics engine.  Supports both OpenGL and Direct3D</td>
<td>This seemed more focused on game and hobby applications than the others, which were more industrial and scientific. It supports both Direct3D and OpenGL, and it seemed less immediate because of this. It enjoys an enthusiastic user community and strong reputation.</td>
</tr>
<tr>
<td>Blender</td>
<td><a href="http://www.blender.org">http://www.blender.org</a></td>
<td>Blender is an open-source 3D animation package; it also has a game engine component</td>
<td>Blender is a quirky piece of software with an unusual interface. Its game engine component has been unsupported at times, but potentially it is quite a powerful environment, allowing a transition from building models, to gaming and interactivity.</td>
</tr>
</tbody>
</table>
Bibliography

Cited Works


Bibliography


Bibliography


Bibliography


Bibliography


Bibliography


Ware, George W. *Reviews of Environmental Contamination and Toxicology*. New York: Springer-Verlag, 2002.

Bibliography


Bibliography

Works not Cited


Bibliography


