Contents

Chapter 1
The Painting Fool
Stories from Building an Automated Painter

Simon Colton

Abstract The Painting Fool is software that we hope will one day be taken seriously a creative artist in its own right. This aim is being pursued as an Artificial Intelligence project, with the hope that the technical difficulties overcome along the way will lead to new and improved generic AI techniques. It is also being pursued as a sociological project, where the effect of software which might be deemed as creative is tested in the art world and the wider public. In this chapter, we summarise our progress so far in The Painting Fool project. To do this, we first compare and contrast The Painting Fool with software of a similar nature arising from AI and graphics projects. We follow this with a discussion of the guiding principles from Computational Creativity research that we adhere to in building the software. We then describe five projects with The Painting Fool where our aim has been to produce increasingly interesting and culturally valuable pieces of art. We end by discussing the issues raised in building an automated painter, and describe further work and future prospects for the project. By studying both the technical difficulties and sociological issues involved in engineering software for creative purposes, we hope to help usher in a new era where computers routinely act as our creative collaborators, as well as independent and creative artists, musicians, writers, designers, engineers and scientists, and contribute in meaningful and interesting ways to human culture.

1.1 Introduction

Computational creativity is the term used to describe the sub-field of Artificial Intelligence research where we study how to build software that exhibits behaviours deemed creative in people. In more practical terms, we investigate how to engi-

Simon Colton
Computational Creativity Group, Department of Computing, Imperial College, 180 Queens Gate, London SW7 2RH, United Kingdom. e-mail: sgc@doc.ic.ac.uk

Fig. 1.1 An example picture from The Painting Fool's *Dance Floor* series.

neer software systems which take on some of the creative responsibility in arts and science projects. This usage of computers in the creative process differs from the majority of ways in which software is used, where the program is a mere tool to enhance human creativity. In contrast, within computational creativity research, we endeavour to build software which is independently creative, either to act as a collaborator with people, or to be an autonomous artist, musician, writer, designer, engineer or scientist.

Within the Computational Creativity Group at Imperial College[1], we engage in various projects where we aim to build software for creative purposes. We work from an Artificial Intelligence perspective, whereby the solutions to problems we encounter while trying to engineer creative behaviour help to improve existing AI techniques, or lead to the invention of new ones. In a major project within the group, we are building *The Painting Fool* program, which we hope will one day be taken seriously as a creative artist in its own right. The project has been ongoing for around 7 years, driven largely by the author, but with input in recent years by PhD students, MSc students and research associates in the group. An example image from one of the most recent projects with The Painting Fool – as described in Section 1.4.3 below – is given in Figure 1.1. As with the HR mathematical invention system (Colton; 2002), which we have developed continuously since 1996, we plan to work on The Painting Fool in perpetuity, that is, for as long as it takes to satisfy the intellectual challenge of building an autonomously creative system.

[1] The web pages for which are here: http://www.doc.ic.ac.uk/ccg.

In one respect, we have fairly low standards: an automated painter doesn't have to produce art at the level of a great master, an esteemed professional, an art school graduate or even a talented amateur artist. At least to start with, The Painting Fool's art has been rather naive and of little cultural interest, but as we progress with the project, we hope the value of the artworks it produces will increase. In another respect, however, we have fairly high standards: to be called a painter, our software must simulate a range of both cognitive and physical behaviours common to human painters. Such behaviours naturally include practical aspects such as the simulation of making paint strokes on a canvas. However, we are also intent on simulating such behaviours as the critical appraisal of one's own work and that of others; cultural and art-historical awareness; the ability to express ideas and moods through scene composition, choice of art materials and painting style; and the ability to innovate in artistic processes.

For some in the art world, there is a discernible resistance to using a computer in art practice, and this is naturally heightened when mention is made of the software acting as a creative collaborator or an independent artist. It is therefore an interesting challenge to gain some level of acceptance for AI-based art producing software within mainstream artistic communities. One problem has been that the majority of artworks produced by software with some level of autonomy have limited appeal and the pieces largely exist for decorative purposes. For instance, once any aesthetic pleasure and possibly some awe at the power of modern computing has worn off, it is difficult to have a conversation (in the cerebral, rather than the literal sense) with an image of a fractal, or indeed many of the generative artworks that artists and engineers regularly produce. Also, as a community of computational creativity researchers, there has been the assumption (or perhaps hope) that the artefacts produced by our software – poems, pictures, theorems, musical compositions, and so on – will speak for themselves. In certain creative domains, this may be the case. For instance, it is likely that people will laugh at a funny joke regardless of how it was conceived. However, in other domains, especially the visual arts, there is a higher level of interpretation required for consumption of the artefacts. In such domains, we have somewhat neglected the framing of the artefacts being produced by our systems. Such framing includes providing various contexts for the work, offering digestible descriptions of how it was produced, and making aesthetic, utilitarian or cultural arguments about the value of the work. Only with this extra information can we expect audiences to fully appreciate the value of the artefacts produced autonomously by computers via more interesting, more informed, conversations.

With The Painting Fool, we are building a system that addresses the shortcomings described above. In particular, we are overcoming technical challenges to get the software to produce more stimulating artworks which encourage viewers to engage their mental faculties in new and interesting ways. These techniques include new ways to construct the paintings, in terms of scene composition, choice of art materials, painting styles, etc. In addition, they also include new ways to frame the paintings, in terms of providing text about the artworks, putting them into context, etc. We are pioneering computational creativity approaches which adhere to principles designed to not only produce culturally valuable artefacts, but also to frame

them in a way which makes them more interesting to audiences. We have argued in favour of these principles from a philosophical viewpoint in (Colton; 2008b), and we have used them practically in the construction of The Painting Fool. Having said that, we are still a long way off achieving our goal, and The Painting Fool is not yet producing pictures of particularly high cultural value, or framing its work in interesting ways.

The purpose of this chapter is to present the current state of The Painting Fool project, to discuss some of the cultural issues raised, and to describe some ways in which the project will continue. It is beyond the scope of this chapter to give a full technical specification of the software, which runs to around 150,000 lines of Java code, and relies on numerous other pieces of software. In place of these details, we refer to various technical papers where the functionality of the software is described at length. In section 1.2, we present our work in some artistic, engineering and scientific contexts. By placing our work in these contexts, in addition to studying state of the art practices in computational creativity and through discussions with numerous people about building an automated painter, we have put together a number of guiding principles which we adhere to in building and framing our software. These guiding principles are outlined in section 1.3. To best describe our progress so far with The Painting Fool project, in section 1.4 we present the motivation, cultural and social issues, technical difficulties and research results for a number of projects carried out within this research programme. In section 1.5, we describe future projects that we intend to pursue towards the goal of building our automated painter, and getting it accepted into society. We conclude in section 1.6 by summarising the issues which arise from the project and calling for collaboration on this project.

1.2 The Painting Fool in Context

Our personal preference is to think of computing as an engineering discipline which uses both scientific and artistic methodologies to evaluate the computer programs we design and engineer. Theoretical scientific methodologies are often employed in order to have the ideas for software in the first place, and then experimental scientific methodologies are employed to test the performance of software in terms of ability, efficiency, reliability, etc. In the visual arts, software is largely employed as enabling tools for artists to produce pieces of art or design. Increasingly, especially in so-called *new media* circles, this has led to software itself being assessed in terms of its cultural and artistic impact. This is most obvious with interactive digital art, where audience members are (hopefully) intellectually stimulated through interaction with software. In addition, video games are increasingly being seen as artistic artefacts, and art students are regularly presenting software, such as novel web browsers, as art objects in their degree shows. It is fairly rare to see computer programs shown in galleries or exhibitions, unless they are interactive art pieces or they generate visual and/or acoustic artworks for visitors. This shouldn't be taken as an indication that general (i.e., non-art producing) software cannot be artistically valuable, because

acceptance of novel media such as software is generally rather slow. For instance, only in 2009 did the Royal Academy in London first accept video installations for its Summer Exhibition.

The visual art software we produce in computational creativity circles largely fits into the mould of art-generating programs. However, there are two important differences which set our programs aside from others in this mould. Firstly, there is the underlying assumption that our software has some creative input to the process. Sometimes, this creative input is in terms of the automatic assessment (and rejection or selection) of artefacts. Alternatively, the input may be in terms of searching a space of artworks which can lead to surprising and interesting results. As a loose rule of thumb, and without wanting to be too exclusive, if the software is not making some kind of decision (whether about assessment and/or avenues of exploration), it is unlikely to be considered to be within the realm of computational creativity. Secondly, computational creativity software can itself produce new programs, hence it can act at a meta-level. Sometimes, this meta-level is not obvious, for instance, the majority of evolutionary art systems produce programs (genotypes) which are compiled or interpreted and executed to produce artworks (phenotypes). However, the user is normally only ever shown the phenotypes, and in this sense the evolutionary software can be seen as an interactive art installation which enables the user to produce aesthetically pleasing artworks. Occasionally, the meta-level is more obvious, for instance in Colton and Browne (2009) we evolved simple art-based games, where the user could click on a spirograph being drawn in order to affect the drawing process. If the user clicked correctly, the spirograph would be drawn to look like a given one, which provided the game playing challenge. In this instance, therefore, our evolutionary software was employed to produce new interactive programs for artistic and playful purposes.

The Painting Fool is a generative art program with decision making abilities that place it in the realm of computational creativity. It is definitively not a tool for artists to use, and hence we do not make it available as such. Rather, we see it as a fledgling artist that is being trained to act increasingly more creatively. In this sense, our automated painter most closely resembles the *AARON* program written by Harold Cohen and described in McCorduck (1991). This is a very well known system that has been developed over 40 years to produce distinctive figurative art, according to Cohen's unique guidance at both generative and aesthetic levels. It is over-simplistic to say that AARON has been developed to paint in the style of Cohen, as he has been influenced himself by feedback from the software, so the process has been somewhat circular. However, it is fair to say that AARON has not been developed to be independent of Cohen. Taken together as a package, Cohen and AARON represent one of the biggest success stories of AI art, both in terms of critical appraisal, acceptance (to a certain level) by the art world and sales to collectors and galleries. Part of this success can be attributed to AARON being seen as creative by various groups of people (although only in a guarded way by Cohen). This is because it invents scenes from imagination, i.e., each scene that it paints is different, and doesn't rely on digital images, etc. Moreover, the scenes are figurative rather than abstract, hence the software uses information about the way the world works, which is not often the

case with generative computer art. Cohen has used AARON to raise issues about the nature of software in art, which has further increased the interest in the artworks it produces. For instance, he ends (Cohen; 1995) by asking:

> "If what AARON is making is not art, what is it exactly, and in what ways, other than its origin, does it differ from the 'real thing?' If it is not thinking, what exactly is it doing?"

The main difference between The Painting Fool and AARON is in terms of the range of artistic abilities that the two pieces of software have. For instance, the range of scene types that can be painted by AARON have differed somewhat over the years, but are largely limited to figurative scenes involving multiple people, pot plants and tables in a room. We discuss later how, when properly trained, The Painting Fool can produce pieces which depict a wide variety of scenes, including ones similar to those produced by AARON. The notion of training highlights another difference between the two systems. To the best of our knowledge, AARON has only ever been programmed/trained by Cohen, and that is not likely to change. In contrast, again as described below, we have built a teaching interface to The Painting Fool which enables artists, designers and anyone else to train the software in all aspects of its processing, from the way in which it analyses digital photographs to the way in which it constructs and paints scenes. We hope that allowing the software to be trained by artists will ultimately enable it to produce more varied and culturally valuable pieces. In addition to this, we have enabled the software to interact with online information sources, such as Google and Flickr and social networking sites such as Facebook and Twitter, as described below and in (Krzeczkowska et al.; 2010). Again, the hope is that the software can be trained to harness this information to produce more culturally interesting paintings.

Future versions of The Painting Fool will be further distinguished from AARON by their ability to critically appraise their own work, and that of others. Cohen provides aesthetic guidance to AARON by programming it to generate pieces in a certain style. However, he has not supplied it with any critical ability to judge the value of the pieces it produces – and ultimately, Cohen acts as curator/collaborator by accepting and rejecting pieces produced by the system. In contrast, not only do we plan for The Painting Fool to use critical judgement to guide its processing, we also plan for it to invent and defend its own aesthetic criteria to use within these judgements. For instance, it will be difficult, but not impossible to use machine vision techniques to put its own work into art-historical context, and appraise its pieces in terms of references (or lack thereof) to existing works of art. In addition, we plan a *committee splitting* exercise, whereby we use crowd sourcing technologies such as Facebook apps to enable members of the public to score pieces produced by The Painting Fool. The software will derive aesthetic measures via machine learning techniques applied to the results of this crowd-sourcing activity. However, we will attempt to avoid so-called "creativity by committee" by enabling The Painting Fool to concentrate on those pictures which are liked and disliked by the crowd in equal measures. In this way, its first learned aesthetic will hopefully be able to tell whether a piece it produces is divisive or not, which is a start. We plan to enable the software to invent and employ various aesthetic measures in a similar fashion.

It's our intention for the art produced by The Painting Fool to cause audiences to engage their mental faculties, and not just to think about the fact that the pieces were computer generated (although we advocate full disclosure of how the software produces its artwork). This will be achieved through the production of intrinsically interesting work, which includes emotional content, interesting juxtapositions, social commentary, and so on. A level of audience engagement will also be made through the software framing its pieces in various art-historical and cultural contexts, and providing titles and wall-text to this extent. There are already art generating programs which achieve a good level of engagement with audiences, some of which are described in other chapters of this book. Indeed, there are many different kinds of conversations one can have with generative art pieces. For instance, in some of the pieces produced by the *NEvAr* evolutionary art system described in (Machado and Cardoso; n.d.), rather than being driven by the user, the software uses built-in fitness functions to search for art generating programs. When viewing these pieces, one might be tempted to try and determine what the fitness function was and how it is expressed in the pieces, in much the same way that one might try and work out the aesthetic considerations going through a human painter's mind when they painted their works. Concentrating on evolutionary art, other projects have appealed to the (un)natural world to evoke feelings in audiences. In particular, often the fact that generated creatures (by for instance, Sims (1994)) and flora and fauna (by for instance, McCormack (n.d.)) look so similar, yet dissimilar to real examples of the natural world can lead to feelings of other-worldliness. McCormack's work on evolutionary decay takes this further, via appeal to the art-historical mainstay of mortality. Similarly, the software in the Mutator project as originally described by Todd and Latham (1992) produces organic forms which can be unnerving – possibly because of a similar effect to the well-known uncanny valley effect in video games, where automated non-player characters get too close to being human-realistic, causing an uncanny, uneasy feeling in many people.

All of these approaches produce works which are thought-provoking independently of their evolutionary genesis, and there are numerous other generative art projects which produce interesting and culturally relevant artworks, with Romero and Machado (1997) providing a good starting point for further reading. However, authors such as Galanter (2010) point out that often the most interesting aspect of evolutionary artworks is the process which went into producing them. This follows a long line of art movements where the principle innovation has been the production process (e.g., impressionism: painting en plein air to catch fleeting light conditions; pointillism: painting with complimentary dots of paint, as per colour theories, to produce more vivid pieces, etc.). We certainly advocate providing a description of the processes at work when software produces pieces of art. However, our position is that how the work is produced should form only one part of the framing of generative artworks, and they would be culturally more important if the pieces themselves offered a reason for audiences to think about certain issues, or if they invoked certain feelings or moods.

Another context within which our project can be seen is that of the graphics subfield of *Non-Photorealistic Rendering* (NPR). Here, the emphasis is on producing

software which simulates natural media such as paints, pencils, canvases, pastels, and their usage in paint strokes, filling regions of colour, etc. Much of the pioneering work in this area has ended up in software such as Adobe Illustrator, which give artists new digital tools and media to work with. As a good example, James Faure-Walker (2006) mixes simulated paint with real paint in his art practice. NPR software is designed along solid software engineering and Human-Computer Interaction lines to be useful and reliable tools for artists and designers. Moreover, given that the consumers of such software are largely within the creative industries (and hence possibly perceived to be worried about creative software taking their over some of their responsibilities), there have occasionally been mistakes of judgement from NPR experts keen to downplay claims of creativity in their software. In particular, in a standard NPR textbook, Strothotte and Schlechtweg state that:

> "Simulating artistic techniques means also simulating human thinking and reasoning, especially creative thinking. This is impossible to do using algorithms or information processing systems" (Strothotte and Schlechtweg; 2002, page 113).

It is difficult to tell whether this statement is denying the subfield of computational creativity research, or the entire field of Artificial Intelligence. In any case, the statement attempts to reinforce the myth that creativity is beyond scientific study, which is one of the main issues addressed within creativity studies and computational creativity research in particular, as addressed most vocally by Boden (2003).

Other NPR researchers are more enlightened, however, and supply their techniques with more intelligent abilities. For instance, with the saliency-adaptive painting research, Collomosse and Hall (2006) enabled their NPR system to determine the most important regions in an image using an evolutionary search. This enabled the production of painterly renditions of digital images with special attention paid to the most salient regions, which is more in line with the way in which painters understand the content of the pictures they are painting.

To summarise our placing of The Painting Fool project into various contexts, we observe that it is generative art software which has evolutionary search and non-photorealistic rendering abilities, in addition to the ability to construct scenes in a similar fashion to AARON. It is being engineered and further trained to transcend most generative art projects by addressing higher level artistic behaviours such as critical ability and cultural awareness. As such, it is designed not as a tool for artists to employ, but rather as a creative collaborator, or even an independent artist.

1.3 Guiding Principles

The building of creative systems requires overcoming numerous technical problems both of a general nature and in the particular domain within which the system works. Given that the results arising from such systems are ultimately for general consumption, building AI systems to create culturally interesting artefacts also requires a certain amount of framing and promotion. Over the years, we have developed the

following seven principles to which we try to adhere when building creative software and which we hope may be useful frames of reference for other people building similar systems. They stand as a paradigm within which to build, test, employ and promote the output of creative software.

1.3.1 Ever decreasing circles

We start with the observation that it is much easier to put together artificially intelligent systems if we have something concrete to work towards, especially when there is a general and workable theory of human intelligence to guide us. This has led to a somewhat unspoken notion in computational creativity that we should be looking towards research about human creativity for guidance on how to get computers to behave creatively. While such natural creativity research influences computational creativity research to some extent, our efforts in building creative software similarily influences our understanding of creativity in general. So, we shouldn't wait for philosophers, psychologists, cognitive scientists or anyone else to give us a workable impression of what creativity is. We should embrace the fact that we are actually undertaking research into creativity in general, not just computer creativity. Hence, we should continue to build software which undertakes creative tasks, we should study these systems, and we should help in the goal of understanding creativity in general. In this way, there will be ever-decreasing circles of research where we influence the understanding of natural creativity, then it influences our research, and so on until we pinpoint and understand the main issues of creativity in both artificial and natural forms.

1.3.2 Paradigms lost

The problem solving paradigm in AI research is well established and dominant. It dictates that when an intelligent task needs to be automated, we immediately ask the same questions: does it involve proving something; does it involve generalising a pattern; does it involve putting together a plan, etc. If it is possible to answer yes to any of these questions, then the task is pigeonholed forever as a theorem proving problem, or a machine learning problem, or a planning problem, etc. This often means that the original aim of the task is lost, because only researchers in the designated area will work on automating approaches with respect to particular problems. As people, we don't solve the problem of writing a sonata, or painting a picture, or penning a poem. Rather, we keep in mind the whole picture throughout, and while we surely solve problems along the way, problem solving is not our goal. The main push to resurrect the lost paradigm of artefact generation – whereby the production of culturally important artefacts is the point of the exercise – is coming from the computational creativity community, and we should educate the next generation

of AI researchers in the need to embrace entire intelligent tasks which lead to the production of beautiful, interesting and valuable artefacts.

1.3.3 The whole is more than a sum of the parts

We have observed that some of the more interesting pieces of software which undertake creative tasks are those where multiple systems have been combined. Certainly, the only systems we've personally built which might be called creative involve at least two pieces of AI software written for different tasks being brought together so that the whole is more than a sum of the parts. There is still a tendency to re-implement techniques to fit into the workflow of a creative system rather than investigating how existing AI software could be incorporated. Sub-tasks within creative systems are often being achieved sub-optimally with a bespoke system for, say, generalisation that could be solved with an off-the-shelf machine learning system. Similarly, we have seen some deductive tasks performed using a forward-chaining approach that would be laughed at by automated reasoning researchers. We should assume that anyone who has built software and made it available for others would be very pleased to see it used in a creative setting, and this might help attract more people to computational creativity. It takes real effort to build systems which rely on other people's software, but the benefits are much greater, as the power and flexibility of the software vastly increase.

1.3.4 Climbing the meta-mountain

Software is mostly a tool for humans to use, and until we can convince people that computer programs can act autonomously for creative tasks, the general impression will remain that software is of no more use to society than, say, a microwave. The main problem is that, even within computational creativity circles, we still build software that is intended to be used by, or at least guided by, us. A common way in which this manifests itself is that we let the software have some autonomy in the production of artefacts (which may involve the software assessing artefacts, for instance), but we retain overall creative responsibility by choosing which artefacts to present to the world. Moreover, a criticism that people often level at so-called creative software is that it has no purpose. That is, if people didn't run the software, analyse its output and publish the results, then nothing would happen – which is not a good sign that the software is autonomously creative. This is a valid criticism. However, it is one that we can manage by repeatedly asking ourselves: what am I using the software for now? Once we identify why we are using the software, we can take a step back, and write code that allows the software to use itself for the same purpose. We call this process climbing the meta-mountain. If we can repeatedly ask, answer, and code software to take on increasing amounts of creative responsibility,

it will eventually climb a meta-mountain, and begin to create autonomously for a purpose, with little or no human involvement.

1.3.5 The creativity tripod

In many domains, in particular the visual arts, how an artefact is produced is very much taken into account when people assess the value of the artefact. This leads to a genuine, and understandable, bias towards human artefacts over computer generated ones, and this feeling is impervious to any Turing test which demonstrates that people cannot tell the difference between human and computer generated artefacts when they are presented out of context. This isn't a fatal problem, though, as long as we are happy to manage the public's impression of how our software works. In our many dealings with (well meaning) critics of computational creativity, we have found that the main criticisms levelled at programs purporting to be creative is that they lack skill, or they lack appreciation, or they lack imagination. We should therefore manage these misconceptions by describing our software along these dimensions. Moreover, we should regularly ask ourselves: if I were to describe my software using this supporting tripod of creativity terms, where would the weakest link be? In identifying and addressing such weak links, we can build better software both in terms of what it is able to achieve practically, and what it appears to be doing.

Managing the perception of creativity in software is as important as building more intelligent algorithms in domains where cultural, contextual and historical precedents play an important role. Hence, if you have software which doesn't appreciate its own work, or the work of others, or its subject material, etc., then you should write code which achieves this. If you have software which isn't particularly inventive, then you should implement some routines which could be described as imaginative, and so on. Using this tripod, we've managed to devise a base-line test for creativity in software which is defensible. We suppose that the software is regularly producing artefacts with certain behaviours being exhibited simultaneously or in sequence during the production process. If from these behaviours, one could genuinely be described as skilful, one could be described as appreciative, and one could be described as imaginative, then we argue that the software should be described as creative. There are two caveats here: firstly, this is not a prescription for creativity in people; secondly, this is a base-line test, i.e., it doesn't mean that the software is highly creative. Indeed, it is our responsibility to keep adding skilful, appreciative and imaginative behaviours so that the software is perceived as increasingly creative.

1.3.6 Beauty is in the mind of the beholder

By changing the sentence "Beauty is in the eye of the beholder" to the one above, we want to emphasise that when people appreciate/buy artwork, the actual look of the finished piece is only one thing they take into consideration. Other things that occupy their mind may include details about the artist and their previous work, other pieces of art owned by the art appreciator, or they have seen in museums, whether the artwork will increase in value, etc. Most importantly, as argued in the previous subsection, people tend to take into account how a piece of art was produced when assessing the finished product. If no information pertaining to the production of an artwork is available, then people can fall back on general knowledge about the struggle artists have in taming paint on a canvas, and can try and reverse-engineer the specifics of this from the paint strokes exhibited. These fall-backs are not available for software generated artefacts, as most people have little idea about how software works. Turing-test style experiments may seem attractive because it shows some level of success if the artefacts being generated by a creative system are vaguely comparable to those produced by people. However, computers are not humans, and this fact should be celebrated, rather than hidden through Turing tests. In the visual arts in particular, Turing-style tests ignore process and promote pastiche, both of which are done at great peril.

We argue that computational creativity researchers should be loud and proud about the fact that our software is generating artefacts that humans might be physically able to produce, but might not have thought to actually bring into being. Many people have asked why The Painting Fool produces artworks that look like they might have been hand drawn/painted. It does seem like we are missing an opportunity to produce pieces that humans can't produce, thus supplementing global art production, rather than producing more of what people are already good at producing. This is a valid point, which we address to some extent in Section 1.4.5 below. However, as pointed out by a reviewer of this chapter, wasps and ants can produce patterns which can't be produced by people. Moreover, simpler machines like telescopes and simple software like fractal image generators can do the same. Automatically producing images which can't be produced by people is easy, but not enough to demonstrate creativity. Automatically producing images which look like they could have been produced by people (because they include figurative details, messages, intriguing references, skilful flourishes, etc.), but – importantly – have not yet been produced by people because no-one has so far thought to do so, is much more difficult, because it requires creativity on the part of the software. It is for this reason that The Painting Fool continues to produce hand-drawn looking images. No self-respecting art school graduate wants to be mistaken for another artist, and should be horrified if they were mixed up with Picasso or Monet in a blind test. We should write software which similarly wants to produce uniquely interesting works of art which are not confused with anyone else's, whether human or computer.

Another reason we believe we should not hide the fact that the artefacts are generated by a computer, is because this kind of deception can set the computer up for a fall. For instance, imagine a Turing-tester saying: "And so, I can now reveal that

these are the paintings produced by a recent art school graduate, and these are the paintings produced by... a convicted murderer". While this example may be a little crass, it makes the point: by stating that the aim is to produce artefacts which look like they might have been created by a person, it explicitly lowers the value of the artefacts produced by computer. By using Turing-style tests, we are seemingly admitting that pastiche is all that we aim for. At best, this shows that we don't understand one of the fundamental purposes of creative endeavours, which is to produce something interesting which no-one has produced before. In many domains, there is no right or wrong, there is only subjective impression, public opinion and the values of influential people in that domain. As there is no reason why we can't change public opinion, there is no reason why we should compare our computer generated artefacts to those produced by people. We can change the mind of the beholder to more appreciate the value of the artefacts produced by our software, and in trying to do so, we can learn a lot about the general perception of creativity in society.

Taking all the above arguments into consideration, we advocate non-blind comparison tests of human and computer art, where full disclosure of the processes behind the production of each piece is given. It is not imperative that the software generated artefacts look like they could be physically human-produced, but it might help people to appreciate them. In such non-blind tests, if art lovers choose to buy computer generated art as much as human art, because the pieces they buy stimulate their mind as well as their eye, we can claim real progress in computational creativity.

1.3.7 Good art changes your mind

It is perhaps not useful to delve here into the debate about what is and what isn't art. However, it is difficult to argue against the fact that some of the best scientific discoveries force us to think more about the Universe we inhabit, and some of the best works of art, music, and literature were explicitly designed to make their audience engage their brains more than usual. Sometimes, the artworks are designed to make most people engage their brains in roughly the same way, other times the artworks are meant to be interpreted in many different ways. Sometimes, the purpose is to engage people on a cognitive level, other times the purpose is to engage them on an emotional level. Given this, our software should produce artefacts with the explicit purpose of making the human audience think more. This can be achieved in a number of ways (disguise, commentary, narrative, abstraction, juxtaposition, etc.), and some of these are easier to achieve than others.

More than any other aspect of computational creativity research, this sets us apart from researchers in other areas of AI. In these other areas, the point of the exercise is to write software to think for us. In computational creativity research, however, the point of the exercise is to write software to make people think more. This helps in the argument against people who are worried about automation encroaching on intellectual life: in fact, in our version of an AI-enhanced future, our software might

force us to think more rather than less. However, there are also powerful works of art which emphasise the phenomenological experience of the work, or which are best appreciated through types of meditation, such as the paintings of Mark Rothko. Hence, as well as hoping to increase mental activity with some of the artefacts that our software produces – which would literally change peoples' minds, whether in terms of an long-held opinion or temporary feeling – we should also hope to change the state of the minds of audience members. In either case, it is clear that, if we want computational creativity software to have impact on people, it should have individual and collective models of the minds of audience members.

1.4 Illustrative Projects

Our purpose with The Painting Fool project is to build an automated painter which is one day taken seriously as a creative artist in its own right. In order to do this, we have developed a roadmap based on the notion of climbing a meta-mountain as described above. That is, we have specified a sequence of very broad areas within which to research and implement improved versions of the software, by asking the question "what does a painter do?" and answering as follows:

1. Makes marks on a canvas
2. Represents objects and scenes pictorially
3. Paints scenes in different styles
4. Chooses styles in a meaningful way
5. Paints new scenes from imagination
6. Invents scenes for a purpose
7. Learns and progresses as an artist

Naturally, this is a very subjective and quite naive breakdown of painterly progression, and is not intended for anything other than directing components within our research programme. As such, it serves its purpose well, and as we will see each component described below fits into one of the parts of this roadmap and contributes to the overall goal of producing an independent artist. For each component, our overriding aim is to implement more sophisticated versions of the software. However, determining what represents an improved program is often one of the more difficult aspects of the project, and we use both engineering standards and feedback from people who view the artworks produced to assess the level of success of each component. Hence, for the majority of the components, we present details of the motivations and aims; some implementation details; results from scientific testing of the software; and a gallery of images arising from running the software, along with some commentary on the value of the images, based on the feedback we have received.

Fig. 1.2 Four different segmenting styles.

1.4.1 Non-Photorealistic Rendering

Starting with the notion of an artist simply making marks on a canvas, we implemented abilities for the software to simulate natural media such as pens, pencils, pastels, paints, brushes, papers and canvases. These tools allow the system to create the basis of a artwork, for example applying paint strokes on a canvas, or making pencil marks on paper. To employ these simulations in useful ways, we implemented standard machine vision techniques to enable the software to identify regions of colour in a digital image, i.e., image segmentation. This led to a graphics pipeline whereby an image is first segmented into a set of paint regions, and then each region is filled and/or outlined a series of times in possibly differing styles. To enhance this pipeline, we enabled layers of different segmentations to be defined for possibly different areas of the original digital image (for instance, the user could define a region of the image as containing a person's eyes, and specify that it is segmented into more regions than the other areas, then painted differently). We were careful to ensure that each stage of the pipeline can be user-controlled by a fairly large number of parameters. For instance, image segmenting is controlled by 12 parameters, including the number of segments required, the smallest segment area allowed, the amount of abstraction of the segment regions, whether to allow segments to have holes, etc. In addition, it is possible to map the colours in the segmentation to a set of colours from another palette, for example art deco colours. The four different segmentations of a flower in Figure 1.2 give some indication of the range of segmentations possible via different paramaterisations.

Image segmenting and the simulation of natural media are all standard non-photorealistic rendering techniques, as described in textbooks such as (Strothotte and Schlechtweg; 2002). We differed from the standard approach in one main respect, namely that we didn't implement different methods for different media types. For instance, the simulation of paints is usually treated differently to the simula-

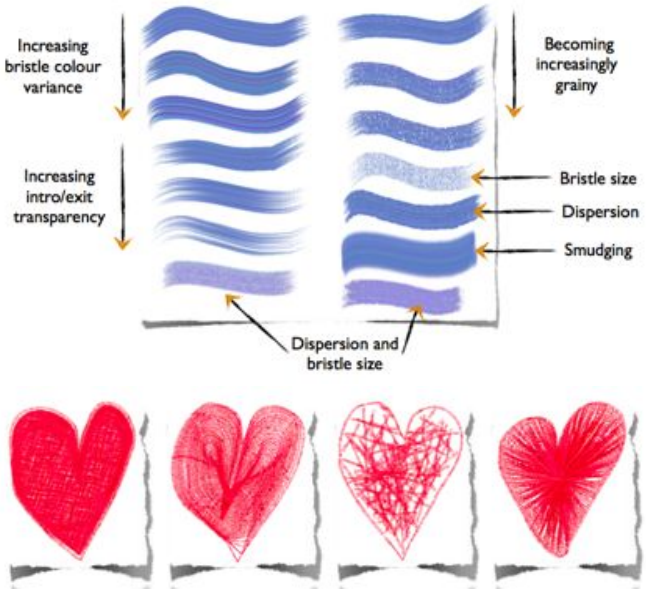

Fig. 1.3 Example paint strokes and region filling mechanisms.

tion of pencils or pastels, etc. Instead, we saw each media type as applying varying amounts of pigment to a fixing medium such as paper or canvas. For instance, pencil strokes could be seen as paint strokes carried out with a very thin brush and a less than usual probability of the pigment sticking to the canvas (which gives the grainy look required). As the individual strokes are only ever used to fill in colour regions, we combined the parameterisation for the individual strokes with the parameterisation of the way in which the strokes were employed. There are 45 parameters controlling the way in which colour regions are rendered, and these include: aspects of the natural media, e.g., brush and bristle size and colour variation; aspects of the individual strokes, e.g., length, taper, curvature; and aspects of the style in which the strokes are used, e.g., ever-decreasing circles versus parallel line fill, number of strokes required to fill a region, etc. The images in Figure 1.3 give a flavour of the different types of strokes and filling mechanisms available, but this is only the tip of the iceberg – many more are possible.

Treating different media types with different models leads to more realistic looking paint strokes. However, for our purposes, treating all the natural media types and their usage as parameterisations of the same method essentially defines a search space of simulations, which has advantages. In particular, there are parts of the search space which fall in-between particular natural simulations, such as paints and pencils. Hence, it was possible to specify ways of filling colour regions with unusual strokes which don't necessarily look like they could be naturally produced. Moreover, we were able to use search mechanisms to find natural media simulations for specific purposes, which we used to discover novel painting styles to enhance

Fig. 1.4 A picture from the city series gallery by The Painting Fool.

emotional content, as described below. Of course, these advantages would still be present if we had separate simulation models for each natural medium, but it would have seriously complicated the search mechanisms.

Full details of The Painting Fool's non-photorealistic rendering capabilities are available in (Colton et al.; 2008). In terms of the wider project, having the ability to turn images into painterly renditions of them enabled us to present some pictures in a group exhibition of computer generated art in 2007. An image from that exhibition is given in figure 1.4. It forms part of a series of eight images of buildings and cityscapes presented in the city series gallery at `www.thepaintingfool.com`.

The exhibition gave us our first platform to introduce the notion of an independent software artist, which enabled us to identify the first of many non-technical issues related to this notion. In particular, the exhibition gained some press and media attention, and predictably led to stories about computers taking over the jobs of people in the arts. In one case, a news team told the story of computer generated art in an TV article prefixed by the phrase: "Is this some kind of hellish nightmare?" This is surely an over-reaction, but it serves to highlight the public's perceived fear over the automation of human abilities, particularly in creative domains such as painting. In response to this, we argue that established artists have more to fear from the latest batch of art school graduates than from computers, because there will always be a premium in art for human involvement.

It does not diminish the potential for computer generated art to engage audiences in meaningful dialogues if we point out that many people appreciate art entirely because of the human aspect: art lovers want to get to grips with the mind and mood of the human behind the artwork. Hence, computer generated art may well occupy different niches to that produced by people, and so there is little to worry about in the automation of painting processes. More interestingly, the news team described above also interviewed Ralph Rugoff – director of the Hayward Gallery in London – and asked for his response to the notion of computer generated art. He pointed out that while software is good at playing games with fixed rules, such as chess, it is less obvious that computer programs can be playful in an artistic sense,

where there are no such rules and where cultural knowledge plays an important role. Moreover, James Faure-Walker (another artist at the exhibition), pointed out that most of the research in non-photorealistic graphics was essentially photograph based, i.e., images are turned into painterly renditions. He added that this is rather a naive approach, and noted that an idea rather than an image should be the motivation for a piece of art. The issues raised by Rugoff and Faure-Walker led us to address the (lack of) imaginative aspects of the software, and ultimately provided the inspiration for the projects described under the scene invention and collage generation sections below.

1.4.2 Emotional Modelling

Human emotion plays an enormous role in the visual arts. Often, paintings are produced in order to convey the emotion of the painter, or to evoke a particular emotion in the viewer. In many cases, an important aspect of appreciating an artwork boils down to understanding the emotions at play. When building an automated painter, we have two choices with respect to emotional modelling. We could simply admit that computers are not human, and therefore any attempt for the software to simulate emotions or model the emotions of viewers would be facile and doomed to failure. In this case, we can emphasise that computer generated paintings can still evoke emotions in viewers without necessarily modelling human emotions, and that there are many other dialogues one can have with a painting other than trying to understand the emotional state of the painter who produced it. Given that we argue in the guiding principles given above that we should celebrate the difference between computers and people, it is certainly a defensible option to ignore emotion. However, this would miss an opportunity to use pioneering work from the field of affective computing, as described in Picard (2002), where software has been built to both simulate and detect human emotions. For this reason, we chose to implement some simple but foundational emotional modelling in The Painting Fool.

We first asked the question of whether we can train the software to paint in different styles, so that it can choose a particular style in order to heighten the emotional content of a painting. Note that this corresponds with part 4 of the meta-mountain described previously, i.e., choosing styles in a meaningful way. We worked on portraits of the actress Audrey Tatou as she portrayed Amélie Poulain in the film *Le Fabuleux Destin d'Amélie Poulain*. This source material seemed appropriate, as the film is largely about the emotional rollercoaster that Amélie finds herself on, and the actress portrays a full range of emotions during the film. Working with 22 stills from the film, we first annotated the images to specify where the facial features were, and then we repeatedly suggested painting styles to The Painting Fool. The descriptions of styles specified the level of abstraction to obtain through the image segmenting; the colour palette to map the regions of colour to; the natural media to simulate while filling/outlining the regions and the brush stroke style to employ while doing so. Largely through trial and error, we derived many of the styles by

hand, by experimenting until the pictures produced were subjectively interesting. In addition to these hand-derived styles, we also enabled the software to randomly generate painting styles. Each time a style – whether randomly generated or derived by us – subjectively heightened the emotion portrayed by the actress in one of the stills, we recorded this fact. In this way, we built up a knowledge base of around 100 mappings of painting styles to emotions, with roughly half the styles provided by us, and the other half randomly generated (but evaluated by us).

Naturally, this tagging exercise was a very subjective endeavour, as all the emotional assessment was undertaken by us. Therefore, in order to gain some feedback about the knowledge base, we built an online gallery of 222 portraits produced from the 22 stills. The gallery is called Amélie's Progress and can be viewed at `www.thepaintingfool.com`. The portraits are arranged from left to right to portray emotions ranging from melancholy on the left to mild euphoria on the right. Sometimes, the emotion portrayed is largely due to the actress, but at other times, the painting style has heightened the emotional content of the piece. Hence, on a number of occasions, we find the same still image painted in different ways on both the left and the right hand sides of the gallery. An image of the entire gallery and some individual portraits are presented in Figure 1.5.

The Amélie's Progress project raises some issues. In particular, we decided to make the web site for The Painting Fool read as if The Painting Fool is a painter discussing their work. This has been mildly divisive, with some people expressing annoyance at the deceit, and others pointing out – as we believe – that if the software is to be taken seriously as an artist in its own right, it cannot be portrayed merely as a tool which we have used to produce pictures. In addition, we chose to enable people working with The Painting Fool to see it paint its pictures stroke by stroke, and we put videos of the construction of 24 of the Amélie portraits onto a video wall, as part of the online gallery. This construction involves the sequential placing of thousands of paint strokes on a canvas. In another area of the web site, there are live demonstrations of paintings being constructed (as a Java Applet, rather than as a video of The Painting Fool at work). We have found that most people appreciate the painting videos, as they promote empathy with the software to some extent. While we know at least an approximation of the painting process for humans, in most cases – especially with the complex mathematical machinations of Photoshop filters – we do not know how software produces painterly images. Hence, seeing each simulated paint stroke applied to the canvas enables viewers to project effort and decision making processes onto the software. We have argued above that the process behind art production is taken into account when people assess the value of pieces of art, and we have much anecdotal evidence to highlight how evaluation of The Painting Fool's pieces is increased after people see the videos of it at work. Of course, this approach was pioneered by Harold Cohen, as it has always been possible to view AARON at work, and AARON was an inspiration for us in this respect. Moreover, in discussions about how software can better frame and promote their own work with Palle Dahlstedt suggested that the artefacts produced by music and visual art systems should contain at least a trace of the construction process

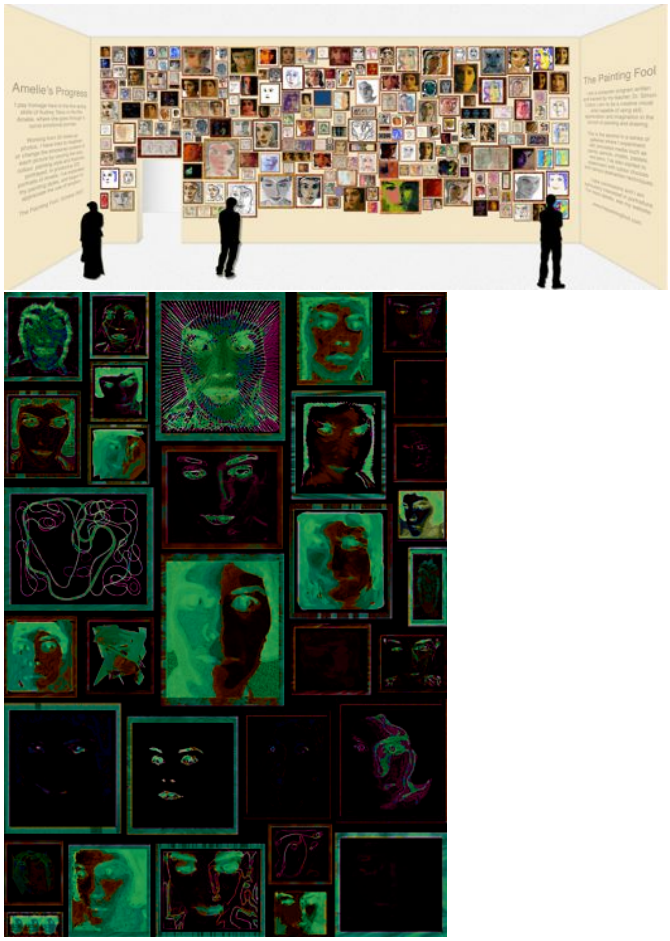

Fig. 1.5 An overview of the Amélie's Progress gallery and some sample portraits from it.

(see Chapter **??**). In simulating paint strokes and showing construction videos, we achieve this with The Painting Fool.

One criticism of most image manipulation software is that it has no appreciation of the images it is manipulating. Hence a Photoshop filter will apply the same techniques to an image of kitten as it would to an image of a skyscraper, which clearly has room for improvement. To address this, and following on from the Amélie project, we addressed the question of whether The Painting Fool can detect emotion in the people it is painting and use this information to produce more appropriate portraits. Detecting emotion in images and videos is a well researched area, and we worked with Maja Pantic and Michel Valstar in order to use their emotion detection software (Valstar and Pantic; 2006), in conjunction with The Painting Fool. The combined system worked as follows: starting with the sitter for a portrait,

Fig. 1.6 Example portraits using styles to heighten (L to R) sadness; happiness; disgust; anger; fear and surprise.

we asked them to express one of six emotions, namely happiness, sadness, fear, surprise, anger or disgust, which was captured in a video of roughly 10 seconds duration. The emotion detection software then identified three things: (i) the apex image, i.e., the still image in the video where the emotion was most expressed (ii) the locations of the facial features in the apex image, and (iii) the emotion expressed by the sitter – with around 80% accuracy, achieved through methods described by Valstar and Pantic (2006). It was a fairly simple matter to enable The Painting Fool to use this information to choose a painting style from its database of mappings from styles to emotions and then paint the apex images, using more detailed strokes on the facial features to produce an acceptable likeness. We found subjectively that the styles for surprise, disgust, sadness and happiness worked fairly well in terms of heightening the emotional content of the portraits, but that the styles for anger and fear did not work particularly well, and better styles for these emotions need to be found. Sample results for portraits in the six styles are given in Figure 1.6.

The combined system was entered for the British Computer Society's annual Machine Intelligence Competition in 2008, where software has to be demonstrated during a 15 minute slot. The audience voted for the Emotionally Aware Painting Fool as demonstrating the biggest advancement towards machine intelligence, and we won the competition. More importantly for The Painting Fool project, we can now argue that the software shows some degree of appreciation when it paints. That is, it appreciates the emotion being expressed by the sitter, and it has an appreciation of the way in which its painting styles can be used to possibly heighten the emotional content of portraits.

1.4.3 Scene Construction

Referring back to the creativity tripod described in the guiding principles above, we note that through the non-photorealistic rendering and the emotional modelling projects, we could claim that the software has both skill and appreciation. Hence, for us to argue in our own terms that the software should be considered creative, we needed to implement some behaviours which might be described as imaginative. To do so, we took further inspiration from Cohen's AARON system, specifically its ability to construct the scenes that it paints. It was our intention to improve upon AARON's scene generation abilities by building a teaching interface to The Painting

Fool which allows people to specify the nature of a generic scene which the software can then produce instantiations of. As described below, we have experimented with numerous techniques in order to provide people with a range of methods with which to train the software. The techniques include AI methods such as evolutionary search and constraint solving approaches; exemplar based methods, where the user teaches the software by example; and third party methods such as context free design grammars for generating parts of scenes.

We describe a scene as a set of objects arranged prior to the production of a painterly rendition. This could be the arrangement of objects for a still life, the orchestration of people for a photograph, or the invention of a cityscape, etc. In practical terms, this entails the generation of a segmentation prior to it being rendered with simulated paints and pencils. This problem is most naturally split into firstly the generation of the overall placement of elements within a scene – for instance the positions of trees in a landscape – and secondly the generation of the individual scene elements – the trees themselves, composed of segments for their trunks, their leaves, and so on. While this split is appealing, we did not develop separate techniques for each aspect. Instead, we implemented a layering system whereby each segment of one segmentation can be replaced by potentially multiple segments repeatedly, and any segmentation generation technique can be used to generate the substitutions. This adds much power, and shown in the example pictures below, allows for the specification of a range of different scene types.

Our first exploration of scene generation techniques involved evolving the placement of scene elements according to a user-defined fitness function. Working with the cityscape scene of the tip of Manhattan as an *inspiring example* (in the words of Ritchie (2007)), we defined a fitness function based on seven correlations between the parameters defining a rectangle, with a set of rectangles forming the cityscape scene. For instance, we specified that there needed to be a positive correlation between a building's height and width, so that the rectangles retained the correction proportions. We similarly specified that the distance of a rectangle from the centre of the scene should be negatively correlated with the rectangle's height, width and saturation, so that buildings on the left and right of the scene were smaller and less saturated, leading to a depth effect. The genome of the individuals were the list of rectangles making up the scene. Crossover was achieved by swapping contiguous sublists, i.e., splitting the genomes of parents into two at the same point and producing a child by taking the lefthand sublist from one parent and the righthand sublist from the other parent (and vice-versa for another child). Mutation was achieved by randomly choosing an individual with a particular probability, the mutation rate, for alteration. This alteration involved changing one aspect of its nature, such as position, shape or colour.

We experimented with one-point and two-point crossover, and with various mutation rates, population sizes and number of generations, until we found an evolutionary setup which efficiently produced scenes that looked like the tip of Manhattan (Colton; 2008a). We turned each rectangle into a segment of a segmentation, and The Painting Fool was able to use these invented scenes as the subject of some pictures. Moreover, we used the same techniques to evolve the placement of flowers

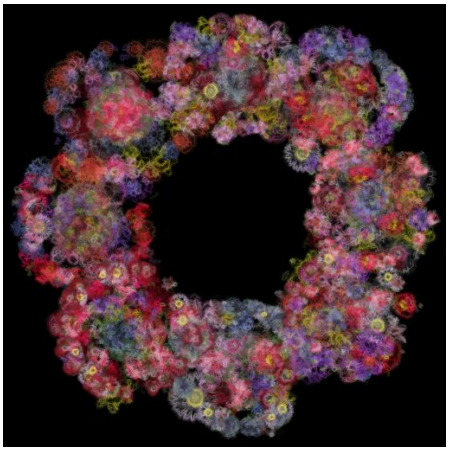

Fig. 1.7 A flower arrangement piece from the "Pencils, Pastels and Paint" gallery by The Painting Fool.

in a wreath effect, with the rectangle position holders replaced by segmentations of flowers. When rendered with pencil and pastel effects, these arrangements became two of the pieces in the "Pencils, Pastels and Paint" permanent exhibition, as described at `www.thepaintingfool.com`, with an example given in Figure 1.7.

In an attempt to climb the meta-mountain somewhat, we realised that in defining the fitness function, we had ultimately performed mathematical theory formation. This suggested that we could employ our HR mathematical discovery system (Colton; 2002), to invent fitness functions in our place. Using the same parameters required to define the original correlations (rectangle width, height, hue, saturation, brightness, and co-ordinates) as background information, and by implementing a new concept formation technique involving correlations, we enabled HR to invent new fitness functions as weighted sums of correlations over the parameters. For each fitness function, we calculated the fitness of 100 randomly generated scenes. If the average fitness was greater than 0.8, then it was likely that optimal fitness was too easy to achieve, and if it was less than 0.4 then it was likely that there were some contradictions in the fitness function. Hence, we only accepted fitness functions with an average for the 100 random scenes of between 0.4 and 0.8.

For each of ten acceptable invented fitness functions, we evolved a scene to maximise the fitness, and on each occasion, the scenes exhibited visually discernable properties. Moreover, two of the scenes genuinely surprised us, because the fitness functions had driven the search towards scenes which we didn't expect. In particular, for one fitness function, the fittest scene involved clumping together the rectangles in three separate centres (scene G in Figure 1.8), and for another fitness function, the fittest scene had buildings placed on top of each other (scene C), which was not expected at all. The ten scenes arising from the fitness functions are given in Figure 1.8, along with two randomly generated scenes, for comparison (R1 and R2). This approach to the invention and deployment of fitness functions is described fully in Colton (2008a). It raises the issue of software defining, employing and defending its own aesthetic considerations, something we will come back to in future work.

Fig. 1.8 Ten scenes generated for different invented fitness functions and two randomly generated scenes.

It also highlights one of the accepted tenets of computational creativity research, which is the notion that creative software should surprise its programmers.

Specifying correlation-based fitness functions for evolutionary scene generation worked well, but it had two main drawbacks: (i) for artistic purposes, sometimes the scene must fully adhere to some constraints, yet there is no guarantee that it will be possible to evolve a scene scoring 100% for fitness (ii) specifying a fitness function is not always a particularly natural thing to do and it would be better if someone using The Painting Fool's teaching interface were able to express their desires for a scene in a visual manner. To address these issues, we investigated the usage of constraint solving, whereby the requirements for a scene, or an element within a scene, are expressed by dragging, scaling and changing the colour of a set of rectangles. Following this, the constraints expressed in the scene are induced and translated into a constraint satisfaction problem (CSP, as described by Abdennadher and Fruhwirth (2003)) and then the CSP is solved to give one or more instances of the scene which differ from the one defined by the user, while still satisfying all the required constraints. Full details of the implementation and our experimentation with it are given in Colton (2008c).

In summary, the user is able to visually express constraints involving: (a) the ranges of properties of rectangles, such as their co-ordinates, colours, dimensions, etc., (b) co-linearity of points on rectangles, (c) propositional notions describing pairs of rectangles, such as the constraint that if the width of rectangle 1 is greater than that of rectangle 2, then its height should also be greater, (d) correlations between the properties of a rectangle, and (e) constraints specifying overlap or disjointness of pairs of rectangles. The software then induces the constraints and asks the user to check whether each one is there by design, or has risen co-incidentally, in which case it can be deleted. At this stage, the user can also describe which aspects of the scene are to be generated randomly, for instance they can specify that the X co-ordinate of the rectangles should be chosen randomly from within a range. The

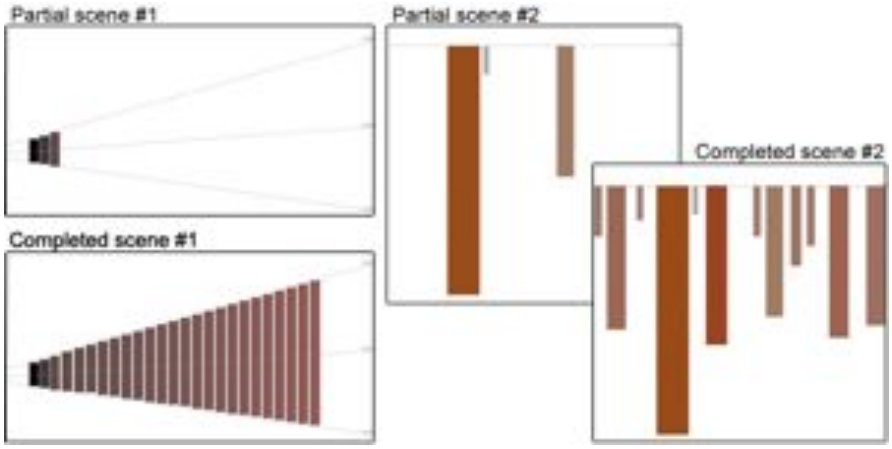

Fig. 1.9 Partial scenes provided by the user and completed scenes produced by solving the induced constraint satisfaction problem.

constraints are then interpreted as a CSP for the Sicstus CLPFD solver (Carlsson et al.; 1997). The variables of the CSP are the co-ordinates, dimensions and colour values of a set of rectangles, with the set size also specified by the user. Hence a solution to the CSP represents a scene of rectangles, and if a random element has been introduced, each scene will be different. To test the constraint solving approach, we worked with an inspiring example of trees in a forest, which ultimately led to the "PresidENTS gallery" as described below. The guiding scene and an example generated scene are provided in Figure 1.9 for two constructions.

Unfortunately, for scenes of 10 or more elements, we found that the constraint solver could take a prohibitively long time to find a perfect solution, and hence we re-integrated the evolutionary approach so that the fitness function could be defined as the number of singletons or pairs of scene elements adhering to the constraints. This means that the visual specification of the scene constraints can be used with a faster evolutionary approach, although the resulting scene may not fully satisfy all the constraints (which in scenes with more elements may actually be desirable).

To supplement the constraint-based and evolutionary approaches, we wanted the teaching interface to enable the user to simply draw an example of the scene or scene element that they wanted to specify, and for the software to use this as an exemplar in order to generate similar looking examples. To do this, we implemented a drawing interface that records key anchor points of each line drawn by a user. The anchor points are recorded as variables rather than fixed values, so they can vary within ranges in order to produce similar looking shapes in a scene. Additionally, we allow the user to specify the hue, saturation and brightness ranges within which the colour of each shape can vary, and to specify allowable linear transformations (such as translations and rotations) and non-linear transformations (such as perspective warping) that entire shapes, or even the entire scene can be subjected to.

To further supplement the scene generation abilities of the teaching interface, we integrated the CFDG generative art software[2], and our own evolutionary art software (Hull and Colton; 2007). The former system is able to generate representational and abstract artworks by using context free design grammars, and there are thousands of grammars available for use in art projects. The latter system is able to generate abstract art forms in a number of styles, including pixel-based (similar to fractals), particle based, and spirograph based (Colton and Browne; 2009). Finally, the sixth scene generation method available within the teaching interface is to take a digital image and turn it into a segmentation, as described in the non-photorealistic rendering section above. We further enabled the image to be filtered before it is segmented, as per our *Filter Feast* software (Torres et al.; 2008).

In addition to a screen for each of the segmentation generation methods described above, the teaching interface has a screen to describe how the different methods are to be used in layers to form the overall scene. It also has a screen to describe how the different elements of the scene are to be rendered using NPR techniques and a screen to describe how to generate paint dance animations (see below). The teaching interface is currently in beta development. While it is not yet ready for general usage, it is possible to define and render scenes. Given that we hope to attract other people to use the software to define their own pictures, it was important to provide example projects which produce interesting images. This has led us to the production of a series of galleries, collectively entitled: "Ever so Slightly…". The rather strange name recognises the fact that it is not feasible that anyone will project a great deal of imagination onto software which is able to produce novel scenes using a template provided by people, but it may be possible to project a slight amount of imagination onto the software, and this is our aim.

There are currently four galleries in the series, named: "PresidENTS", "Fish Fingers", "After AARON", and "Dance Floor". An example from the "Dance Floor" series has been given in Figure 1.1, and we give examples from the others in Figure 1.10. The titles of the first two are fairly awful puns which reflect their content, and we won't spoil the fun of working out the wordplay here (see `www.thepaintingfool.com`). The fourth one reflects the influence that Cohen's AARON system has had on the scene generation aspects of The Painting Fool – as we see in the images given below, the pictures strongly reference the contents of the pictures produced by AARON, although we did not try to capture it's visual style. Note that the human figures were produced by context free design grammars, and the abstract images on the walls of the room are similarly produced. The gradient effect of the ceiling and floor used a constraints approach to the generation of rectangles which were filled in using simulated pencils.

[2] Available at `www.contextfreeart.org`

Fig. 1.10 Pictures from the "PresidENTS", "Fish Fingers" and "After AARON" galleries in The Painting Fool's "Ever So Slightly…" exhibition.

1.4.4 Collage Generation

The pictures produced in the "Ever So Slightly…" series represent a step in the right direction towards imaginative behaviour. However, looking at the meta-mountain we have described for The Painting Fool, the software needs to construct scenes for a purpose. Moreover, while the paintings in the series may be amusing and mildly thought-provoking as simple word/art puzzles, they are certainly not the most provocative of works. One aspect of the human painting process that is rarely simulated in computer art is the ability to construct paintings to convey a particular message within a cultural context. We looked at using text and image resources from the internet as source materials for the production of artwork that might have a cultural impact (Krzeczkowska; 2009). In particular, the software began by downloading headline news stories from the websites of the Guardian newspaper and other news sources. Using text extraction software (El-Hage; 2009), based on the TextRank algorithm (Mihalcea and Tarau; 2004), the most important nouns were extracted from the text of the news story. These were then used as keywords for searches in image repositories, including Google images and Flickr. The resulting images were juxtaposed in a collage which was turned into a segmentation, and the non-photorealistic rendering software from The Painting Fool was used to produce a painterly rendition of the subject material. With the exception of the text extraction software, the process was largely devoid of AI techniques, and this is something we plan to work on. However, the results were often remarkably salient. As an example, one morning the software downloaded the lead story from the Guardian, which was covering the war in Afghanistan, and used images from Flickr to illustrate it. The final collage was quite poignant, as it contained a juxtaposition of a fighter plane, an explosion, a family with a small baby, a girl in ethnic headwear, and – upon close inspection – a field of war graves. The collage is given in Figure 1.11.

In Krzeczkowska et al. (2010), we used this project to raise issues of intent in generative software. Usually, the intent for a piece is supplied by a human user, possibly through the expression of an aesthetic judgement and/or tailoring the content to fit the intent. However, with the Afghanistan collage, we were not users of the software in the traditional sense. Firstly, the software ran as a timed batch process, hence we didn't hit the start button. Secondly, we had no idea that the software would find a story about war, and thirdly, we had no idea which keywords it would

Fig. 1.11 Collage produced in response to a news story about the war in Afghanistan.

extract or which images it would retrieve for the collage. Hence, it is difficult to say that we supplied much intentionality for the collage, even though the painting does achieve a purpose, which is to force the viewer to think about the Afghanistan war. We argue that it is possible to say that the software provided some of the intent, but we acknowledge that this is controversial. In fact, it seems clear that five parties contributed intent in the construction of the Afghanistan collage: (i) the programmer, by enabling the software to access the left-leaning, largely anti-war Guardian newspaper (ii) the software, through its processing, the most intelligent aspect of which was the extraction of the keywords (iii) the writer of the original article, through the expression of his/her opinions in print (iv) individual audience members who have their own opinions forming a context within which the collages are judged, and (v) the Flickr users whose images were downloaded to use in the collage, by tagging many negative images such as explosions and fields of graves with the kinds of neutral words that were extracted from the newspaper article, such as "Afghanistan", "troops" and "British".

We also used the collage generation project to raise the issue of playfulness in the software, as the collages would often contain strange additions, such as an image of Frank Lloyd-Wright's *Falling Water building* being included in a collage arising from a news story about the England cricket team (see Krzeczkowska et al. (2010) for an explanation of how this happened). We don't claim that the word "playful" should be used to describe the software, but it does show potential for this kind of behaviour.

1.4.5 Paint Dances

In many respects, it is an odd choice to build an automated artist that simulates traditional media such as paints and brushes. It would seem to be missing an opportunity for the software to invent its own medium of expression and exploit that. In future, we would have no problem with the software performing such medium invention, and we would see that as a very creative act. However, we argue that while the software is in its early stages it is more important for its behaviour to be understood in traditional artistic terms, so that its creativity can be more easily appreciated. In particular, as described in Section 1.3.6, we want the software to produce paintings that look like they could have been physically produced by a human, but simultaneously look like they would not have been painted by a person because they are so innovative in technique and in substance.

Notwithstanding this notion, we wanted to follow an opportunity for the software to work in a new medium, albeit one which is not far removed from painting, namely, *paint dances*. We define a paint dance as an animation of paint, pencil, or pastel strokes moving around a canvas in such a way that the strokes occasionally come together to produce recognisable subject material. We worked with portraits, in particular our subject material was images of the attendees of the 2009 Dagstuhl seminar on computational creativity. The technical difficulties involved are discussed in Colton (2010). In summary, to achieve the paint dances, we first implemented a way to tell which pairs of strokes from two different paintings were most closely matched. Following this, from the 60,000 pencil strokes used in the 32 portraits of the Dagstuhl attendees, we used a K-means clustering method to extract just enough generic strokes to paint each picture in such a way that the fidelity would remain high enough for a likeness to be maintained. The final technical hurdle was to write software to perform the animations by moving and rotating the paint strokes in such a way as to come together at the right time to achieve a portrait. We have so far completed two paint dances: "meeting of minds", where pairs of pencil portraits are shown together, with the strokes meeting in the centre of the picture as they move to form two new portraits, and "eye to eye", where each painted portrait is formed individually, with spare paint strokes orbiting the scene. The images in Figure 1.12 show the stills from a transition in both pieces. We plan to complete a triptych of video installations with a third paint dance entitled winds of change which animates pastel portraits being blown around.

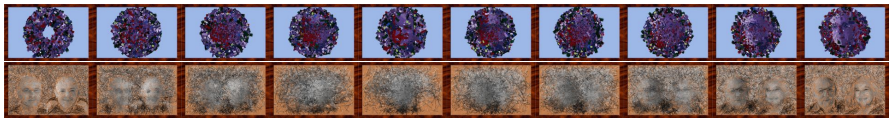

Fig. 1.12 A series of stills from the "Meeting of Minds" and the "Eye to Eye" paint dances produced by The Painting Fool.

This project raises the issue of software undertaking tasks that people cannot perform – in this case, the calculation of where to place thousands of strokes at thousands of time steps. As artists have been finding out since the advent of the computing age, getting software to undertake tasks beyond their capacity can enhance the space of possibilities in creative projects. Moreover, in an age of personalisation, such personalised healthcare, personalised entertainment, etc., the only way to achieve personalised artistry will be to engage computers to produce artworks for people on an individual level. However, we note that the aesthetic decisions for the paint dance project were still undertaken by us, and not the software. Hence, in line with our guideline of climbing the meta-mountain, the next stage for this project will be to enable the software to invent animation techniques and make aesthetic judgements about what to apply while composing the paint dances.

1.5 Future Directions

We tend to undertake projects within quite large research programmes over a lengthy period of time, whereby multi-faceted intelligent systems are built by developing and combining our own techniques with existing AI systems (adhering to the principle that the whole is usually more than the sum of its parts). Ultimately, our aim is for the software we build to exhibit a range of intelligent behaviours while generating culturally interesting artefacts, all within a computational creativity setting. As an example, the programme of research to build an automated mathematical theory formation system has been ongoing for fifteen years and has resulted in the HR system which has been the basis of four PhD projects, and the subject of more than 100 research papers. In this light, it is important to note that The Painting Fool is very much a work in progress, and we are not claiming that it should be taken seriously as an independently creative artist yet (and even if or when we do make that claim, it will be in reference to an artist of a very low ability, at least to start with). In order to discuss future directions for The Painting Fool project, we address our current progress with respect to the guiding principles mentioned above, highlight areas where these principles can be used to assess the system negatively, and suggest ways forward.

With respect to the skill, appreciation and imagination requirements of the creativity tripod, it is easy to argue that the software is able to simulate the kinds of physical skills that are required of a painter. In addition, with the ability to produce paint dances, we can claim that the software has skills not possessed by human painters. There are numerous additional physical skills that we would like to simulate. For instance, the ability to produce smooth colour gradients through paint strokes is something that would certainly enhance the quality of the pieces produced by the software, and other physical simulations such as the use of a palette knife, the ability to spatter paint, etc., would all add value.

The software is more lacking in appreciative and imaginative behaviours than in skilful behaviours. We have argued that with the emotional modelling projects, the

software is exhibiting some level of appreciation of its subject material and its painting styles. The fact that The Painting Fool cannot assess its own artworks and those of others against various aesthetic considerations is a major gap in its abilities. We have implemented abilities for the software to calculate objective measures of abstract art, for instance the location of symmetries, the distribution of colours, regions of high and low texture, etc. However, it is difficult to imagine training the software to appreciate works of art without essentially training it in a singularly subjective way (i.e., to become a "mini-me" for someone). In such circumstances, it would be difficult to argue against the software simply being an extension of the programmer, which we clearly want to avoid. An alternative approach is to build on the project to use mathematical theory formation to invent fitness functions, as described above. Rather than inventing a single fitness function, we hope to show that it is possible for software not only to invent more wide ranging aesthetic considerations, but also adhere to them, change them and discuss and possibly defend them within cultural contexts.

One aspect of this may involve getting feedback from online audiences, which will be used to tailor the image construction processes. However, as mentioned in Section 1.2, we are keen to avoid creativity by committee, which could lead to the software producing very bland pieces that do not offend anyone. Instead, we propose to use a committee splitting process, by which The Painting Fool will judge the impact that its pieces have on people, and choose to develop further only those techniques that led to pictures which split opinion, i.e., those which people really liked or really hated. Enabling the software to work at an aesthetic level will also involve endowing it with art-historical and cultural knowledge to allow it to place its work in context and to learn from the work of others. We are in discussions with artists and art educators about the best way to do this. In addition, we will draw on texts about creativity in both human and computer art, such as Boden (2010).

With the scene generation and collage generation abilities, we make the claim that the software could be considered very slightly imaginative, and we aim to build on these foundations. Firstly, once the teaching interface is finished, we will deploy it by asking numerous people from academia, the arts, and from the creative industries to train the software. The payoff for people using the tool will be the production of pictures which hopefully characterise the ideas they had in mind and would paint if they were using more traditional methods. The payoff for The Painting Fool project will be a collection of potentially hundreds of scene descriptions. We plan to use these in order to perform meta-level searches for novel scenes in a more playful way than is currently possible. An important aspect of the teaching interface is the tagging of information, which is passed from screen to screen in order to cross-reference material for use in the overall scene construction. Hence, the software will in essence be taught about the visual representation of real-world objects and scenes (in addition, of course, to imaginary ones). We hope to build models which can subvert this information in playful and productive ways, building meaningful scenes different to those it was given.

We also intend to extend the collage generation approach described above, whereby online resources are employed as art materials. To this end, we have

begun construction of a computational creativity collective, available at: www.doc.ic.ac/ccg/collective. The collective currently contains individual processes which perform creative, analytical, and information retrieval tasks, along with mashups, which combine the processes in order to generate artefacts of cultural interest. For instance, the project whereby news stories are turned into collages described above is modelled in the collective as a mashup of five processes which retrieve news stories, extract text, retrieve images and construct collages. The collective currently has processes which can link to Google, Flickr, the BBC, LastFM, Twitter and numerous other online sources of information. Our plans for the collective are ambitious: we hope to attract researchers from various areas of computing including graphics, natural language processing, computer music and audio, and artificial intelligence to upload their research systems to expand the collective.

Systems built for computational creativity purposes such as The Painting Fool are beginning to have abilities of note in their particular domains of expertise, but rarely are they combined in order to increase the cultural value of their output. Hence we plan to paint pictures using the text produced by story generators like the Mexica system (Perez y Perez; 2007) as input, and there is no reason why the pictures produced couldn't be used, for instance, as input to an audio generation system. This example highlights the masterplan for the collective, which is to have the output of one system continually consumed as the input to another system, thus providing a framework for experimentation with control mechanisms. In particular, the first mechanism we will implement will be based on Global Workspace Architectures, (Baars; 1988), as per the PhD work of Charnley (2010). It is our hope that the increase in complexity of processing, coupled with the ability to access culturally important information from online resources will lead to more thought-provoking artefacts being generated.

1.6 Conclusions

The Science Museum in London once exhibited some interesting machines made from Meccano which were able to perform fairly complex differential analysis calculations. As these machines were built in the 1930s, the Meccano magazine from June 1934 speculated about the future in an editorial article entitled: "Are Thinking Machines Possible?" (Anon; June 1934). They couldn't have possibly known the impact that the computing age would have on society, but they were already certain about one thing: at the very end of the article, the author states that:

> "Truly creative thinking of course will always remain beyond the power of any machine."

At this stage in the development of modern computing, it is neither a foregone conclusion that we will never see truly creative machines, nor is it obvious that we will one day be working alongside creative individuals which happen to be computers. It is our job as computational creativity researchers to investigate the possibilities for creative software, but we do not underestimate the difficulty of engineering

such systems, and we do not underestimate the difficulties we will face in getting such software accepted on equal terms in society. We have described the overall aim of The Painting Fool project and some of the components we've completed along the way in order to climb a meta-mountain. The next stages will involve enabling the software to learn and develop as a creative painter, and this will raise further issues. One litmus test for progress, or even completion of the project, will be when The Painting Fool starts producing meaningful and thought-provoking artworks that other people like, but we – as authors of the software – do not like. In such circumstances, it will be difficult to argue that the software is merely an extension of ourselves.

The project has always been driven by feedback from people around some of the issues that we have raised here, and we always welcome collaboration in this respect. It seems that creativity in software – and perhaps in people – is usually marked negatively. That is, while there is no sufficient set of behaviours that a computer program must exhibit in order to be deemed creative, there is a necessary set of behaviours that it must exhibit to avoid the label of being uncreative. By adhering to the guiding principles described above in undertaking projects with The Painting Fool, we hope to manage people's perceptions of creativity, most obviously through (i) the notion of climbing the meta-mountain, whereby we describe the ways in which the creative responsibilities we have as programmers and users have been bestowed upon the software, and (ii) the notion of the creativity tripod, whereby we describe The Painting Fool's behaviours in terms of the skills it has, the appreciation that it exhibits and the imagination it exercises. It is our hope that one day people will have to admit that The Painting Fool is creative because they can no longer think of a good reason why it is not.

Acknowledgements We would like to thank the organisers and participants of the 2009 Dagstuhl seminar on computational creativity, for their very interesting discussions, debates and performances, and for permission to use their images in the paint dances. We would also like to thank the Dagstuhl staff for their efforts in making the event very enjoyable. The anonymous reviewers for this chapter provided some excellent food for thought with relation to the arguments that we put forward. These comments have greatly enhanced our understanding of the issues, and have led to a much improved chapter. Many members of the Computational Creativity community have expressed support and provided much input to The Painting Fool project, for which we are most grateful. We owe a great deal of gratitude to the many collaborators who have contributed time and expertise on The Painting Fool and related projects. These include Anna Krzeczkowska, Jenni Munroe, Charlotte Philippe, Azalea Raad, Maja Pantic, Fai Greeve, Michel Valstar, John Charnley, Michael Cook, Shafeen Tejani, Pedro Torres, Stephen Clark, and Stefan Rüger. We would like to lovingly thank Sandra Ashworth for all her amazing support during the long years spent building The Painting Fool, and to apologise for the long years to come doing more of the same.

References

Abdennadher, S. and Fruhwirth, T. (2003). *Essentials of Constraint Programming*, Springer.
Anon (June 1934). Are thinking machines possible?, *Meccano Magazine* .

Baars, B. (1988). *A Cognitive Theory of Consciousness*, Cambridge University Press.
Boden, M. (2003). *The Creative Mind: Myths and Mechanisms (second edition)*, Routledge.
Boden, M. (2010). *Creativity and Art. Three Roads to Success*, Oxford University Press.
Carlsson, M., Ottosson, G. and Carlson, B. (1997). An open-ended finite domain constraint solver, *Proceedings of Programming Languages: Implementations, Logics, and Programs*.
Charnley, J. (2010). *A Global Workspace Framework for Combined Reasoning*, PhD thesis, Department of Computing, Imperial College, London, UK.
Cohen, H. (1995). The further exploits of AARON, painter, *Stanford Humanities Review* **4(2)**.
Collomosse, J. and Hall, P. (2006). Salience-adaptive painterly rendering using genetic search, *International Journal on Artificial Intelligence Tools (IJAIT)* **15(4)**: 551–576.
Colton, S. (2002). *Automated Theory Formation in Pure Mathematics*, Springer-Verlag.
Colton, S. (2008a). Automatic invention of fitness functions, with application to scene generation, *Proceedings of the EvoMusArt Workshop*.
Colton, S. (2008b). Creativity versus the perception of creativity in computational systems, *Proceedings of the AAAI Spring Symposium on Creative Systems*.
Colton, S. (2008c). Experiments in constraint-based automated scene generation, *Proceedings of the 5th International Joint Workshop on Computational Creativity*.
Colton, S. (2010). Stroke matching for paint dances, *Proceedings of Computational Aesthetics*.
Colton, S. and Browne, C. (2009). Evolving simple art-based games, *Proceedings of the EvoGames workshop*.
Colton, S., Valstar, M. and Pantic, M. (2008). Emotionally aware automated portrait painting, *Proceedings of the 3rd International Conference on Digital Interactive Media in Entertainment and Arts*.
El-Hage, J. (2009). *Linguistic analysis for The Painting Fool*, Master's thesis, The Computer Laboratory, University of Cambridge, UK.
Faure-Walker, J. (2006). *Painting the Digital River*, Prentice Hall.
Galanter, P. (2010). The problem with evolutionary art is..., *Proceedings of the EvoMusArt Workshop*.
Hull, M. and Colton, S. (2007). Towards a general framework for program generation in creative domains, *Proceedings of the 4th International Joint Workshop on Computational Creativity*.
Krzeczkowska, A. (2009). *Automated collage generation from text*, Master's thesis, Department of Computing, Imperial College, London, UK.
Krzeczkowska, A., El-Hage, J., Colton, S. and Clark, S. (2010). Automated collage generation - with intent, *Proceedings of the 1st International Conference on Computational Creativity*.
Machado, P. and Cardoso, A. (n.d.). All the truth about NEvAr, *Applied Intelligence* **16(2)**: 101–118.
McCorduck, P. (1991). *AARON's Code: Meta-Art, Artificial Intelligence, and the Work of Harold Cohen*, W.H. Freeman and Company.
McCormack, J. (n.d.). Evolutionary L-systems, *in* L. Barone and Z. Michalewicz (eds), *Design by Evolution: Advances in Evolutionary Design*, Springer, pp. 168–196.
Mihalcea, R. and Tarau, P. (2004). TextRank: Bringing order into texts, *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.
Perez y Perez, R. (2007). Employing emotions to drive plot generation in a computer-based storyteller, *Cognitive Systems Research* **8(2)**: 89–109.
Picard, R. (2002). *Affective Computing*, MIT Press.
Ritchie, G. (2007). Some empirical criteria for attributing creativity to a computer program, *Minds and Machines* **17**: 67–99.
Romero, J. and Machado, P. (eds) (1997). *The Art of Evolution: A Handbook on Evolutionary Art and Music*, Springer.
Sims, K. (1994). Evolving virtual creatures, *Proceedings of SIGGRAPH*.
Strothotte, T. and Schlechtweg, S. (2002). *Non-Photorealistic Computer Graphics*, Morgan Kaufmann.
Todd, S. and Latham, W. (1992). *Evolutionary Art and Computers*, Academic Press.

Torres, P., Colton, S. and Rueger, S. (2008). Experiments in example-based image filter retrieval, *Proceedings of the Cross-Media Workshop*.

Valstar, M. and Pantic, M. (2006). Biologically vs. logic inspired encoding of facial actions and emotions in video, *Proceedings of the IEEE International Conference on Multimedia and Expo*.