Ludic Considerations of Tablet-Based Evo-Art

Simon Colton, Michael Cook and Azalea Raad

Computational Creativity Group, Dept. of Computing, Imperial College, London
http://www.doc.ic.ac.uk/ccg

Abstract. With the introduction of the iPad and similar devices, there is a unique opportunity to build tablet-based evolutionary art software for general consumption, and we describe here the i-ELVIRA iPad application for such purposes. To increase the ludic enjoyment users have with i-ELVIRA, we designed a GUI which gives the user a higher level of control and more efficient feedback than usual for desktop evo-art software. This relies on the efficient delivery of crossover and mutation images which bear an appropriate amount of resemblance to their parent(s). This requirement in turn led to technical difficulties which we resolved via the implementation and experimentation described here.

1 Introduction

While interacting with evolutionary art software can be a very rewarding experience, doing so is not yet a mainstream hobby, with the notable exception of online collaborations such as the Electric Sheep project [2]. The recent proliferation of tablet devices such as the iPad – where an increased emphasis is put on users enjoying their interaction with software – offers a good opportunity to bring evolutionary art to a wider audience. We have built the j-ELVIRA desktop evolutionary art program (which stands for (J)ava (E)volution of (L)udic (V)ariation in (R)esponsive (A)rtworks), as a rational reconstruction of the Avera software [5], in the usual mould of human-centric evo-art software such as NEvAr [6]. Porting j-ELVIRA to the iPad raised more than just issues related to re-programming the software. In particular, the touchscreen interface and the expectation of constant, enjoyable interaction with iPad applications required a new design for the GUI and improvements to the efficiency and quality of image generation. We provide details here of the iPad implementation (i-ELVIRA), including aspects of the interface design and the nature of image generation, in addition to some experiments we have performed to inform our design choices.

In section 2, we describe the particle-based image generation underpinning the ELVIRA systems, and we critique the desktop version from the perspective of user-interaction. This critique led us to design i-ELVIRA in such a way as to increase the *ludic quality* of the software, i.e., how much fun it is to interact with. These design choices led to certain technical difficulties. In particular, we found that images were generated too slowly to achieve the kind of ludic interaction we hoped for, and that often the images were not appropriate, i.e., they were blank, or bore too much/too little resemblance to their parents. In sections 3 and 4, we describe how we improved both the generation speed and quality of the images respectively, including details of some experiments we undertook to assess mutated image fidelity. In section 5, we assess the suitability of i-ELVIRA for mainstream use, and we discuss some future directions for our work.

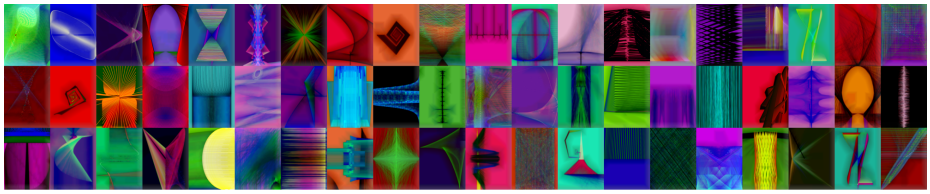
2 A Tablet-Based Interface Design

Particle based image generation schemes have been used to good effect in evolutionary art, for instance in the generation of ricochet compositions [3], and within the neuro-evolution framework described in [4]. In [5], Hull and Colton introduced an image evolution scheme which we have re-implemented into the j-ELVIRA software. The scheme uses six *initialisation trees* to control the nature of P particles, in terms of a sextuplet defining their location and their colour: $\langle x, y, r, g, b, a \rangle$. Over T timesteps, each particle changes colour and position, as controlled by six corresponding *update trees*, and a line is drawn from their previous position to their new one. The genomes of images therefore comprise a background colour which the canvas is filled with initially, and 12 trees which represent mathematical functions that calculate collectively the colour and position of a particle numbered p as it is initialised and as it changes at timestep t. Each update function takes as input the current $\langle x, y, r, g, b, a \rangle$ values of the particle in addition to t and p, all of which may be used in the function calculations. The canvas onto which lines are drawn is defined by the rectangle with corners $(-1, -1)$ and $(1, 1)$. At each timestep, after all the particle lines have been rendered, a Gaussian blur is applied. Each new set of lines is drawn on top of the previously resulting blurred background.

With the default values of $P = 1000$ and $T = 100$, the production of an image requires the plotting of 100,000 lines and 100 blurring operations. In [5], the authors barely explored the variety of images which can be produced, because their implementation was slow due to the drawing (off-canvas) of very long lines. By truncating – if necessary – the start and end points of the lines to be within the canvas rectangle, we achieved a much faster implementation. This allowed greater exploration of the types of images that can be produced, and we have seen a huge variety of images which can evoke perception of: lighting effects, shadowing, depth, texture and painterly effects. Moreover, the images produced often have a non-symmetrical and moderately hand-drawn look, which can add to their appeal. A sample of sixty images produced using this method is given in figure 1, along with an example genome (of 12 flattened trees) and the resulting image. We see that the complexity of the images derives not from the complexity of the individual trees, but rather from the iterative nature of the calculations the trees perform, and the blurring effect, which adds much subtlety to the images.

j-ELVIRA has a user interface similar to that of many evo-art packages:

- To start with, the user waits while a series of randomly generated genomes are used to produce a user-chosen number of images (usually 25).
- The user selects the images they prefer, or sometimes at the start of the session, they tend to select any images which are in any way unusual, as many of the images will be devoid of interest.
- The user chooses to either crossover and/or mutate the selected images. They then wait while the software chooses randomly from the selected individuals and crosses pairs of them and/or mutates them into child genomes from which new images are produced and shown to the user.



Initialisation functions	Update functions
$x(p)_0 = -(0.75/sin(p))/-p$	$x(p)_t = sin(0.25 - p)$
$y(p)_0 = -(p * -0.5) + (p/-0.01)$	$y(p)_t = -sin((x_{t-1}(p) - 0.001) * (a_{t-1}(p)/r_{t-1}(p)))$
$r(p)_0 = cos(cos(sin(-0.001/p)))$	$r(p)_t = sin(x_{t-1}(p)) + 0.75$
$g(p)_0 = -sin(sin(0.001 + 100 * p))$	$g(p)_t = sin((-b_{t-1}(p) - y_{t-1}(p)) * t)$
$b(p)_0 = (-p * p)/(-0.25 * p)$	$b(p)_t = y_{t-1}(p) * -0.5$
$a(p)_0 = sin(-0.01)$	$a(p)_t = cos(t) * 2 * r_{t-1}(p)$

Fig. 1. Top: 60 exemplar images produced with the particles method. Below: a genotype/phenotype pair in terms of the genome functions and the image produced.

When implementing i-ELVIRA in Objective C for the iPad, we referred to Apple's iPad development guidelines, where it is noted that: (a) "People, not apps, should initiate and control actions ... it's usually a mistake for the app to take decision-making away from the user" (b) "People expect immediate feedback when they operate a control", and (c) "When appropriate, add a realistic, physical dimension to your app". We felt that the interaction design for j-Elvira was at odds with guidelines (a) and (b). In particular, by choosing which individuals from the preferred set to crossover/mutate, j-Elvira takes too much control away from the user, thus contradicting guideline (a). Moreover, j-Elvira forces users to wait much too long (measured in minutes) for the production of 25 images before more progress can be made, thus contradicting guideline (b). To address these issues, we designed the iPad interface so that the user explicitly chooses which individual to mutate, and which pairs to crossover. As soon as they have made their choice, the child images are produced immediately. This hands back more control to the user, and they are supplied with feedback as soon as a new image can be generated (measured in seconds). In response to guideline (c), to add a realistic element to the interface, we used metaphors of: a recycling tray for discarding images; a printer/scanner for copying and generating images; and rows of trays into which sets of images can be dragged.

The random generation of images at the start of a session with j-ELVIRA is also unappealing, as it is prone to producing blank/dull images, as all/most of the particle lines are drawn off-canvas. To address this, we generated tens of thousands of images randomly, and chose by hand 1,000 *preset images* from them (60 of which are given in figure 1). These were chosen to maximise the variety of images afforded by the particles method, so that hopefully every user may find a preset which fits their aesthetic preferences. We have performed extensive testing, and we have found that neither large trees, nor more complex functions than cosine and sine, nor more complex programmatic structures such as conditionals lead to more interesting or varied images. Therefore, when generating the preset images, we restricted the random search to trees of size 12 or less, containing only the arithmetic and trigonometric functions and the terminals $\{0, 0.001, 0.01, 0.1, 0.25, 0.5, 0.75, 1, 10, 100\}$ and their negations.

Fig. 2. (i) Opening screen, where old sessions can be loaded (ii) browsing screen, where images are generated and organised into trays (iii) editing screen, where images can be scaled, rotated, translated, cropped and re-rendered.

We provide screenshots of the graphical user interface for i-ELVIRA in figure 2. In overview, the user is presented with a continuous supply of preset images at the top of the screen. The user can drag these images into trays lower on the screen to organise them. The rows of trays add physicality to the design and enable the choosing of any pair of images for crossover. If a user drags one image on top of another, i-ELVIRA immediately crosses over the genomes of the two images to produce four child images, which appear at the top of the screen. Moreover, if the user drags an image to the printer/scanner, i-ELVIRA will immediately start the generation of four mutations of the image. The images are produced at the full resolution of the iPad screen (768×1024 pixels), and by tapping on an image, it expands to fill the whole screen. In this state, the image can be cropped, scaled, rotated and translated, and these transformations are recorded in the image's genome (as a 4×4 matrix). The user can choose to re-render the image to further explore it, which is done by applying the transformation matrix to the position of the particle just before the line between its old (transformed) position, and its new one is rendered.

The interactive nature of the GUI for i-ELVIRA forced two issues. Firstly, users did not expect to wait long for image generation, i.e., they wanted near-immediate gratification in the form of more images in the style of ones they have chosen. Secondly, people expected the images generated in response to their choices to always be *appropriate* to those choices, i.e., that the children of crossed-over or mutated individuals should resemble their parents, but not too much. Unfortunately, our first implementation of i-ELVIRA regularly presented the user with inappropriate images, and took up to 15 seconds to produce each one. In early experiments, we found that this uncertainty in child fidelity and the slow execution times largely ruined the ludic appeal of the software, and hence both issues had to be addressed, as described in the following sections.

3 Efficient Image Rendering

The printer/scanner metaphor helps disguise the time taken to produce images, because as soon as a user has chosen an image for mutation or a pair of images for crossover, an animation of a blank sheet of paper being output holds the user's attention for a few seconds. However, we estimated that any longer than a five second wait for an image to be produced would be detrimental the the user's enjoyment. To calculate the lines which comprise an image, 6 functions have to be called 100,000 times to calculate the position and colour of the particles. Fortunately, our preset genomes contain fairly succinct functions, due to

the fact that we restricted tree size to 12 nodes or fewer. However, we still found that the calculations took prohibitively long: around eight seconds on average. This was because we were representing the functions as trees which were being interpreted at runtime. We chose to flatten the trees into mathematical functions such as those in figure 1 and precompile these into i-Elvira. This dramatically reduced the calculation time for the particles to around half a second on average. Of course, a drawback to precompilation is a reduction in the size of the search space, as new trees cannot be generated at runtime, nor existing ones altered. In particular, the only option for crossover is to swap entire initialisation/update functions of two parents, and for mutation, it is to randomly substitute one or more function with ones from other individuals (i.e., no random generation of trees is possible). However, the trees themselves are fairly small, so there wasn't much scope for crossing over subtrees anyway. Moreover, from the 1000 preset images, we extracted 1798 distinct initialisation functions and 2076 distinct update functions. Hence, given that any initialisation function may be swapped for any other, and likewise for update functions, $1798^6 \times 2076^6 = 2.7 \times 10^{39}$ distinct genomes can be produced, which is more than enough.

Having halved the image generation time through precompilation, we turned to the other major bottleneck: the rendering of the image, i.e., the drawing of the lines and the blurring operation. Apple supplies the 2D iPad *CoreGraphics* graphics library. In our first attempt, we employed CoreGraphics to draw the lines and wrote a per-pixel blurring operation, which changes a pixel's colour to be an average of those in a neighbourhood around it – with a bigger neighbourhood producing a more blurred image. Sadly, this method was too inefficient for our purposes, as it took around 6 seconds to render the image. Hence, we made several improvements to the image generation pipeline in order to optimise the process. The most important of these was using OpenGL ES 1.1, an early mobile version of the popular graphics engine, instead of CoreGraphics. To make the move to OpenGL, we altered the rendering process to employ a vertex-based drawing model, whereby each rendering pass contains a single update to the particles which are described in terms of start and end vertices.

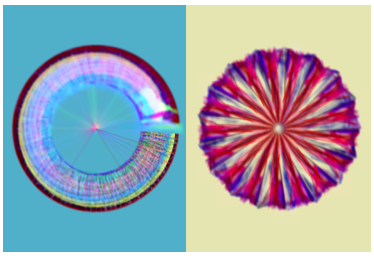
Recall that at each timestep, a blur operation is performed in the image generation process. As OpenGL ES 1.1 does not support pixel shaders, which would have allowed for a per-pixel Gaussian blur to be applied between passes, we instead pass the base image (representing a composite of each rendering stage completed so far) to OpenGL as a texture. After the lines for the current timestep are drawn on top of this texture, a further composite comprising the base image and the new lines is drawn out to another texture using an OpenGL FrameBufferObject. To this second texture, we perform a blur by redrawing the texture object four times, offset by one pixel in the four compass directions, at a reduced opacity. This produces a blurred image without being too costly for OpenGL to draw. The resulting five-layer image is then flattened down to become the base image for the next rendering pass. This new pipeline reduced the rendering time to around 3.5 seconds on average, which is better than the 15 seconds we started with, and within our 5 second ludic limit.

4 Generation of Appropriate Images

The second issue raised by the interactive nature of i-ELVIRA was the disappointment users felt when they were presented with an image which was either blank, or looked different to what they expected (i.e., too similar or dissimilar to its parent). Recall that four mutations of a chosen image are supplied when the user makes a choice, and similarly four offspring are supplied when the user chooses two parents to crossover. Noting that efficiency of image generation is a major issue, we decided not to perform a post-hoc check on image quality, in order to reject an image on grounds of low quality, as this would mean producing another one, and therefore at least doubling the image generation time on occasions. Instead, we concentrated on enabling i-ELVIRA to more reliably generate genomes that would produce appropriate images. Blank or nearly blank images are caused by a lack of lines being drawn on the canvas. One way to avoid the generation of such images altogether is to somehow map the position of each particle at each timestep to somewhere within the canvas. One possibility is to map x and y to their fractional parts, whilst maintaining their parity. Unfortunately, this produces largely uninteresting images, as each line is rendered on the canvas, and many images gain their beauty by having fewer than the total 100,000 lines drawn. For instance, many interesting images exhibit a blurry look, as no lines are drawn on them for the last 10 or 20 timesteps.

However, we did find a number of mappings that regularly produce pleasing images. Two such mappings are given along with sample images produced using them in figure 3. Note that $f(k)$ denotes the fractional part of k, and $\theta_{p,s} = \frac{2\pi(p \bmod s)}{s}$ for a parameter s, which determines the number of segments in the kaleidoscope image (default 17). Given that the images produced by these mappings have appeal, and that the extra processing does not produce a noticeable increase in rendering time, we have enabled i-ELVIRA to generate images using the mappings, and a number denoting which mapping to use is stored in the genome of every generated image. We accordingly added new genomes which use these mappings, to i-ELVIRA's presets. In addition, we looked at the unmapped genotypes which produced the blank images, and we realised that most of them were caused by the update functions for the x and/or y co-ordinates being constant. Hence, we gave i-ELVIRA the ability to avoid producing child genomes through mutation or crossover where either the x or y update function was not dependent on any input value. We found that this drastically reduced the number of blank or nearly blank images to an acceptable level.

People find it difficult to predict what the children of two parent images will look like, and are fairly forgiving when crossover images don't resemble their parents. Indeed, people tend to use the crossover mechanism to search the space of images, rather than to focus on a particular style. We experimented with different crossover mechanisms, until the following emerged as a reliable way to produce child images: given two parent images A and B, child C inherits the background colour of B, and five initialisation functions and four update functions from A, with the missing initialisation and update functions inherited from B. This mechanism works well, except when people crossover two images



Polar mapping: $\begin{pmatrix} x_t(p) \\ y_t(p) \end{pmatrix} \mapsto \begin{pmatrix} f(x_t(p))cos(2\pi f(y_t(p))) \\ f(x_t(p))sin(2\pi f(y_t(p))) \end{pmatrix}$

Kaleidoscope mapping: $\begin{pmatrix} x_t(p) \\ y_t(p) \end{pmatrix} \mapsto \begin{pmatrix} cos(\theta_{p,s}) & sin(\theta_{p,s}) \\ -sin(\theta_{p,s}) & cos(\theta_{p,s}) \end{pmatrix} \begin{pmatrix} f(x_t(p))cos(\frac{2\pi}{s} f(y_t(p))) \\ f(x_t(p))sin(\frac{2\pi}{s} f(y_t(p))) \end{pmatrix}$

Fig. 3. Example images produce by the polar and kaleidoscope particle mappings.

which are themselves the children of a shared parent. In this case, there was a tendency for C to be very similar to A and/or B. Hence, whenever an offspring genome is produced, if the parents share 10 or more functions, the offspring is mutated by swapping one initialisation function for a randomly chosen one, and similarly swapping two update functions. This produces children which vary enough from their parents. In producing four offspring, i-ELVIRA produces two children as above, and two children with the contributions of A and B swapped.

Users of i-ELVIRA tend to be much less forgiving for mutated versions of their chosen images, as mutation is the major way of exerting fine-grained control over image production. Hence users tend to be quite disappointed when a mutated image is too dissimilar or too similar to its parent. Due to the precompiled nature of the functions in i-ELVIRA, the smallest mutation possible is a swapping of a single initialisation function for a randomly chosen one. However, we found that mutating only initialisation functions was prone to producing too many *twin images*, i.e., pairs of siblings in the four produced by i-ELVIRA that look too similar (see the experimental results below). We therefore looked at mutating a single update function, but we found that this could sometimes produce both too dissimilar mutations and too similar ones. Fortunately, we found that both these problems could be managed by enabling i-ELVIRA to perform a *dependency analysis* on the functions in the genomes of images. Looking at table 1, which reiterates the update functions for the genome in figure 1, we see, for example, that the updated alpha value $a(p)_t$ for particle number p at timestep t is dependent on the previous red value of p, namely $r_{t-1}(p)$. However, looking at $r_t(p)$, we see that the updated red value of p is dependent on the previous x co-ordinate of p, namely $x_{t-1}(p)$. Working backwards, we can therefore conclude that output of $a(p)_t$ is dependent on output of the r and x update functions.

For each precompiled update function, we gave i-ELVIRA information about which other functions appear locally, e.g., it is told that $a(p)_t = cos(t)*2*r_{t-1}(p)$ is locally dependent on $r_{t-1}(p)$. We further implemented a routine to work backwards from the local dependencies to determine which variables each function is ultimately dependent on. Hence, in the example in table 1, i-ELVIRA knows that mutating the $r_t(p)$ or $x_t(p)$ update function will also affect the output from the $a_t(p)$ function. In effect, i-ELVIRA can build a dependency matrix such as that in table 1, where a 1 in row w and column c indicates a dependency of the row w function on the column c function. For instance, the 1 in the g row and

	x	y	r	g	b	a		
x	1	0	0	0	0	0	1	$x(p)_t = sin(0.25 - p)$
y	1	1	1	0	0	1	4	$y(p)_t = -sin((x_{t-1}(p) - 0.001) * (a_{t-1}(p)/r_{t-1}(p)))$
r	1	0	1	0	0	0	2	$r(p)_t = sin(x_{t-1}(p)) + 0.75$
g	1	1	1	1	1	1	6	$g(p)_t = sin((-b_{t-1}(p) - y_{t-1}(p)) * t)$
b	1	1	1	0	1	1	5	$b(p)_t = y_{t-1}(p) * -0.5$
a	1	0	1	0	0	1	3	$a(p)_t = cos(t) * 2 * r_{t-1}(p)$
	6	3	5	1	2	3		

Table 1. Dependency analysis for the genome from figure 1.

b column indicates that the update function $g(p)_t$ is dependent on previous b values. Note that we put a 1 in each diagonal position, because each function is dependent on itself (in the sense that mutating a function will naturally alter that function). The row totals indicate how many update functions that row's function is dependent on. The column totals indicate how many update functions depend upon that column's function, and can therefore be used to indicate how disruptive changing that function will be to the location and colour of the particles. We call these numbers the *dependency quotients* for the functions. For instance, the 5 in the r column of table 1 indicates that mutating $r(p)_t$ will effect 5 attributes of each particle, namely their y, r, g, b and a values.

On inspection of a number of mutated individuals where a single update function was swapped for a randomly chosen one, we found that mutating an update function which had a dependency quotient of 1 led to images which were too similar to the original. In contrast, swapping a function which had a dependency quotient of 6 led to a change in every colour and location aspect of each particle, and hence led to a fundamentally different image, i.e., too dissimilar to the original. So, for instance, for the genome in table 1, it would be unwise to mutate either the $x_t(p)$ update function, upon which all the other functions depend, or the $g_t(p)$ update function, upon which no other functions depend. We also analysed a number of other cases where the mutation produced was inappropriate, and used our findings to derive a heuristic mutation mechanism which produces an acceptably high proportion of appropriate mutations.

This mechanism swaps a single update function for a randomly chosen one. It first tries to swap the r, g, b or a function, but will only chose one if it has a dependency quotient between 2 and 5 inclusive. If this fails, it attempts to swap the x or y function, but again only if one has a dependency quotient between 2 and 5 inclusive. If this fails, then an update function is chosen randomly and swapped, ensuring that neither the $x_t(p)$ nor the $y_t(p)$ update function is swapped for a constant function. Moreover, if neither the x nor y function is dependent on the r, g, b or a functions, then either the x or the y initialisation function (chosen randomly) is mutated. Each part the heuristic mechanism was motivated by analysis of the reasons why a set of mutations produced inappropriate images. To determine the value of the heuristic mechanism, we compared it to swapping four, five and six initialisation functions for randomly chosen ones (called the *4-init*, *5-init* and *6-init* methods respectively), and swapping a single update function for a randomly chosen one, with no dependency analysis or constant function checking (called the *1-update* method).

Method	n_b	n_s	n_d	n_t	p_f
Heuristic	6	30	25	29	0.84
1-update	12	34	95	19	0.63
4-init	8	80	68	128	0.66
5-init	12	47	99	114	0.69
6-init	17	33	114	128	0.69

Table 2. Results for 250 sets of 4 images by 5 mutation methods.

We chose 250 of i-ELVIRA's presets randomly, and for each method, we took each preset in turn and produced 4 mutations for it, as this is the number that users of i-ELVIRA are presented with. We performed image analysis on the resulting 1000 mutated images to determine how appropriate they were. We first recorded the number of mutated images which were essentially blank (n_b). We found that two images look similar if they have a similar colour distribution, and/or if they exhibit a similar shape, i.e., the parts of the canvas which are covered by lines for the two images are similar. Given two images i_1 and i_2, we implemented a method to determine the colour distance, $d_c(i_1, i_2)$ of the images in terms if their colour histograms, and a method to determine the shape distance $d_s(i_1, i_2)$, in terms of the positions in a 24×24 grid which are crossed by particle lines. Both methods return a value between 0 and 1, with 0 indicating equal images, and 1 indicating as different as possible images. Analysing pairs of images visually, we settled on two definitions: a pair of images i_1 and i_2 are *too similar* if $min(d_c(i_1, i_2), d_s(i_1, i_2)) < 0.1$, and *too dissimilar* if $max(d_c(i_1, i_2), d_s(i_1, i_2)) > 0.9$.

Within the 1000 images produced for each mutation method, we recorded the number of mutations which were too similar to their parent (n_s) and the number which were too dissimilar (n_d). We also recorded the number of twins produced (n_t). Finally, we recorded the proportion, p_f, of sets of four mutations which contained no inappropriate images (i.e., neither too similar to the parent or eachother, nor too dissimilar to the parent). The results for the five methods are given in table 2. We see that the heuristic method outperforms the other methods in all the metrics, with the exception of the 1-update method producing fewer twins (at the cost of an increased distance between child and parent). Also, as previously mentioned, we see that the mutation methods which alter only initialisation functions all suffer from producing too many twins.

5 Conclusions and Future Work

We have developed the i-ELVIRA evolutionary art application for the iPad, guided by general ludic considerations, which include: enabling constant user-interaction with no periods where the user is waiting for the software to finish processing; avoiding supplying the user with uninteresting or inappropriate images; a natural interaction design which enables the crossing over and mutation of chosen images; an ability for users to customise their artworks and for them to share their creations with others. We are currently finalising i-ELVIRA for distribution, which requires a ludic graphical user interface to the evolution and image generation mechanisms. This has only been made possible because we reduced the image rendering time to 3.5 seconds, and we increased the reliability with which appropriate mutation images are produced. Looking at the results in table

2, we see that the heuristic mutation method delivers a set of four appropriate mutations with an 84% likelihood, which we believe is acceptable for i-ELVIRA. In the context of evolving buildings for a video game, producing artefacts which have an appropriate resemblance to their parents was addressed in [7]. This is a key question in bringing evolutionary art to the general public.

To hopefully improve i-ELVIRA, we will experiment with showing users updates during the rendering process, which might hold their attention (although we have found that for many images, this process can be quite dull, as all the lines are drawn at the start or the end of the process). We will also experiment with different blurring processes to explore different visual styles, and we will enable a search mechanism so that people can find presets similar to ones they like. With all our experimentation, we will undertake extensive user testing to determine the value of the changes we impose. In particular, using i-ELVIRA and j-ELVIRA as research tools, our next step will be to conduct user studies, whereby we try and derive methods for estimating people's reactions to images. Ultimately, we aim to embed machine learning methods into evolutionary art software, so that it can approximate people's aesthetic considerations and use this to deliver better images, as we began to investigate in [1] for image filtering. In the long term, we aim to show that sophisticated user modelling techniques can lead to more enjoyable software such as i-ELVIRA for public consumption, and also be a driving force for machine learning.

Acknowledgements

We would like to thank the anonymous reviewers for their interesting comments. We would also like to thank the Department of Computing at Imperial College, London for funding the implementation of the i-ELVIRA system.

References

1. S Colton, P Torres, J Gow, and P Cairns. Experiments in objet trouvé browsing. In *Proceedings of the 1st International Conference on Computational Creativity*, 2010.
2. S Draves. The electric sheep screen-saver: A case study in aesthetic evolution. In *Proceedings of the EvoMusArt Workshop*, 2005.
3. G Greenfield. Evolved ricochet compositions. In *Proceedings of the EvoMusArt workshop*, 2009.
4. E Hastings, R Guha, and K Stanley. Neat particles: Design, representation, and animation of particle system effects. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, 2007.
5. M Hull and S Colton. Towards a general framework for program generation in creative domains. In *Proceedings of the 4th International Joint Workshop on Computational Creativity*, 2007.
6. P Machado and A Cardoso. NEvAr – the assessment of an evolutionary art tool. In *Proceedings of the AISB00 Symposium on Creative and Cultural Aspects and Applications of AI and Cognitive Science*, 2000.
7. A Martin, A Lim, S Colton, and C Browne. Evolving 3D buildings for the prototype video game Subversion. In *Proceedings of the EvoGames Workshop*, 2010.