Evolutionary Design of Dynamic SwarmScapes

Namrata Khemka
Dept. of Computer Science
University of Calgary, Canada
nkhemka@ucalgary.ca

Scott Novakowski
Dept. of Computer Science
University of Calgary, Canada
snovakow@gmail.com

Gerald Hushlak
Dept. of Art
University of Calgary, Canada
hushlak@ucalgary.ca

Christian Jacob
Dept. of Computer Science
University of Calgary, Canada
cjacob@ucalgary.ca

ABSTRACT

This paper discusses interactive evolutionary algorithms and their application in swarm-based image generation. From an artist's perspective, the computer-generated patterns offer departure points for creative "Imagineering." Input into our evolutionary system comprises of ninety-six parameters that influence the behavior of swarm agents to create emerging "SwarmScapes" over time. This paper demonstrates how an evolutionary approach is used to create swarm-based animations, which at any point in time can be turned into electronic paintings and into high-resolution plots on canvas. As our team consists of three computer scientists and an artist, we also explore the collaborative relationships among the team members, and between the artist and the evolutionary system.

Categories and Subject Descriptors

J.5 [**Computer Applications**]: Arts and Humanities

General Terms

Algorithms, Design, Experimentation, Human Factors

Keywords

Interactive evolutionary art, swarm-based painting, genetic programming, interactive evolution, swarm intelligence

1. INTRODUCTION

The designs produced by natural evolution are extremely diverse. Because of proliferation of visual images through media like television and the Internet, it is inspirational for an artist or designer to utilize a vehicle that creates images outside of the media box. Evolutionary design systems receive increasing attention within various disciplines because they improve productivity, quality, speed, and reduce cost of design [2]. Inspired by and based on the concepts of evolution in Nature, evolutionary systems have symbiotically assisted in the generation of design solutions in domains such as engineering, medicine, social sciences, and art. By utilizing evolutionary algorithms, designers act as 'breeders' by iteratively exploring design options through mutation and recombination of previous design ideas. The superior solutions survive for further breeding and enhancement, while poor solutions disappear from the population. This process is repeated and selectively improved until desirable solutions are established.

Inspired by the capability of evolutionary computation, *Evolvica* was created as an evolutionary design system in the early 1990's involving genetic algorithms, genetic programming, evolution strategies, and evolutionary programming [4]. *Inspirica* is an extension of *Evolvica* and supports evolutionary exploration of design solutions through either automatic fitness evaluation or interactive assessment by a 'breeder'/designer (Figure 1) [7]. *Inspirica* has been applied to various fields including the evolution of implicit surface models of fluid containers, chairs, face masks, and swarm dynamics [8] and architectural idea models [15]. For this paper we have, expanded the *Inspirica* system to generate dynamic, time-varying SwarmScapes.

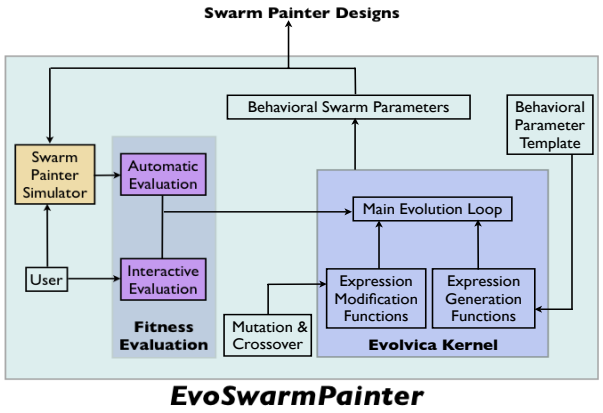


Figure 1: Overview of the *Inspirica* system used for evolving the parameters of the *SwarmPainter* application. Each box in the diagram is a component in the system. Arrows indicate the flow of output from one component into another.

GECCO'08, July 12–16, 2008, Atlanta, Georgia, USA.

The rest of the paper is organized as follows. We present related work in Section 2. Section 3 describes our *SwarmPainter* and *evoSwarmPainter*, the two main modules to generate SwarmScapes. We also describe the representation of the control parameters of a *SwarmPainter* individual and the genetic operators utilized in our evolutionary process. Examples of evolved SwarmScapes and a discussion of their artistic merit are provided in Section 4. Finally, we conclude our work and outline future possibilities in Section 5.

2. RELATED WORK

Interactive Evolutionary Computation (IEC) is being used in domains where the fitness function is not known in advance, or where it is difficult to define the criteria to computationally assess good solutions—such as visual appeal, aesthetics, and attractiveness. Recursive two-dimensional line figures called Biomorphs by Richard Dawkins were the first demonstrations of how interactive evolution can be used for simple parameter evolution [3]. Inspired by Dawkins' work, Latham and Todd created genetically evolved three-dimensional artificial life-like forms that can be considered as "virtual sculptures" [13]. The works of both Dawkins and Latham and Todd had prompted Karl Sims to generate abstract two-dimensional images based on symbolic expressions for computer art, evolved under the guidance of humans [11]. Sims further inspired many computer scientists to use evolution for generating art. Unemi developed Sbart, an image breeding program using selection to evolve images similar to Sims [14]. Rooke used genetic algorithms to evolve fractal art constructed from the Mandelbrot set schemes [10]. Takagi compiled an extensive survey of applied interactive evolutionary computation in areas such as entertainment, data mining, music, and animation [12]. Bentley applied a human-guided fitness evaluator for generating shapes such as drums, flowers, and robots [1]. Some of our team members have also gained considerable experience with interactive SwarmArt installations, utilizing both swarm intelligence and live video processing [5].

The commonality within the above systems is that a human breeder selects preferred designs in order to generate complex images and shapes. Most work referenced above creates static designs, whereas our *SwarmPainter* system is dynamic and generates designs that unfold over time, not unlike developmental models such as L-systems [4] or swarm grammars [6]. The *SwarmPainter* interface offers a simple and interactive application for the user who, ideally, is an artist or designer that is only interested in the actual outcome, but won't engage in any programming or complex parameter tuning necessary to achieve specific outcomes. This is where the evolutionary module, *evoSwarmPainter*, comes into play.

3. PAINTING WITH SWARMS

This paper demonstrates how genetic programming has been used to explore complex pattern formations of a swarm-based painter program operating on a two-dimensional canvas. The stand-alone version of the *SwarmPainter* simulation allows the user to create visuals without having to expend the effort of discrete designs. Once the parameters are established, *SwarmPainter* keeps producing permutations of swarm-generated images (Figure 5) that unfold over time. In Section 3.1 we discuss the technical details of *SwarmPainter*, followed by a discussion on its evolutionary module in Section 3.2.

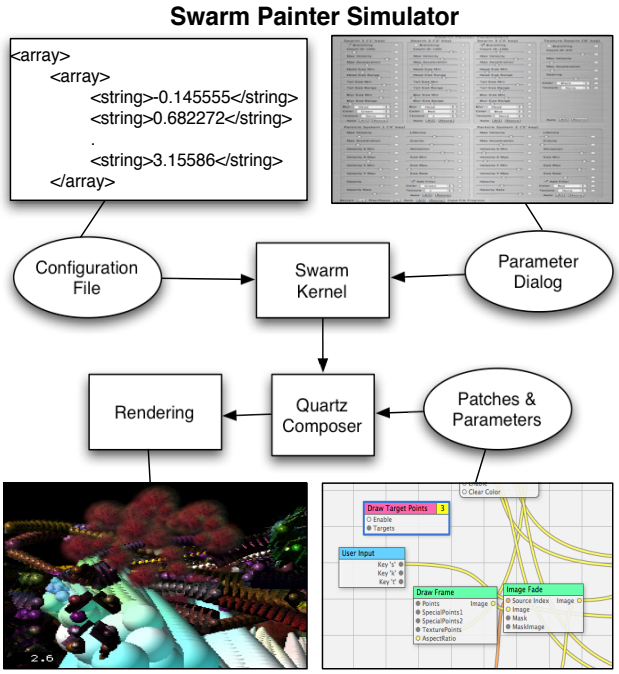


Figure 2: System architecture of *SwarmPainter*: The 96 parameters are entered into the *SwarmPainter* kernel via a configuration file or the parameter dialog. The swarm kernel uses Quartz Composer for rendering the SwarmScapes.

3.1 SwarmPainter

There are three distinct types of agents in *SwarmPainter* that exhibit different behaviors (Fig. 6): (1) swarm agents, S_i, (2) particle agents, P_j, and (3) texture agents, T_k. Each swarm agent is described by fifteen different parameters (Table 1). Twenty parameters portray the particle agents and ten parameters constitute the behavior and appearance of each texture agent. These parameters are read from an XML-encoded configuration file at the start of a *SwarmPainter* simulation. All of these parameters can also be adjusted at any time during a *SwarmPainter* run through a parameter adjustment window (Figure 6a).

3.1.1 Target Points

Painter agents exhibit their own movement patterns determined by their control parameters (Table 1). Additionally, target points can be created, to which swarm agents are attracted. The target point locations can change over time, thus choreographed movement patterns can be imposed on the swarms.

Figure 3 shows an example where four target points are activated one after the other, with a user-specified delay in-between. In this example, the cluster of swarm agents first approaches target point 1, but then gets diverged towards target 2, 3, and 4, thus creating a small swarm sculpture. These target points can either be declared in a configuration file or can be interactively changed by the user at any time

Parameter values of the swarm agents.				
Swarm Parameters	*Description*	$(x_k^{\min}, x_k^{\max})$	*Mutation Radius*	*Mutation Weights*
(x, y) location	(x,y) target which agent urge towards	(-3, 3)	[-2, 2]	15
velocity	velocity of a swarm agent	(0, 100)	[0, 50]	1
acceleration	acceleration of a swarm agent	(0, 10)	[0, 5]	1
color	agents have either a red, green, or blue tendency	(0, 2)	[0, 2]	15
texture	bitmap texture to use for agent rendering	(0, 1)	[0, 1]	1
count	number of swarm agents	(0, 100)	[0, 50]	1
branch	branching paths or continuous paths	(0, 1)	[0, 1]	1
head/tail	minimum agent head/tail size and the range in which head size can fluctuate	(0, 0.3)	[0, 0.2]	1
blur	blur size and range in which the blur size can fluctuate	(0, 0.1)	[0, 0.05]	1

Table 1: Swarm parameters: A selection of parameters that affect the behavioral patterns of the three swarm agents in the *SwarmPainter* simulation are listed. The third column states the minimum and maximum values $(x_k^{\min}, x_k^{\max})$ allowed for each of the parameters. The forth column contains the mutation radii ($\Delta x_k = [\Delta x_k^{\min}, \Delta x_k^{\max}]$). The last column lists the weights for mutating each entry. For the crossover operator only the (x, y) location and color parameters are utilized, with the same weights as the mutation operator. Particle and texture agents have parameters similar to the swarm agents but are omitted for brevity.

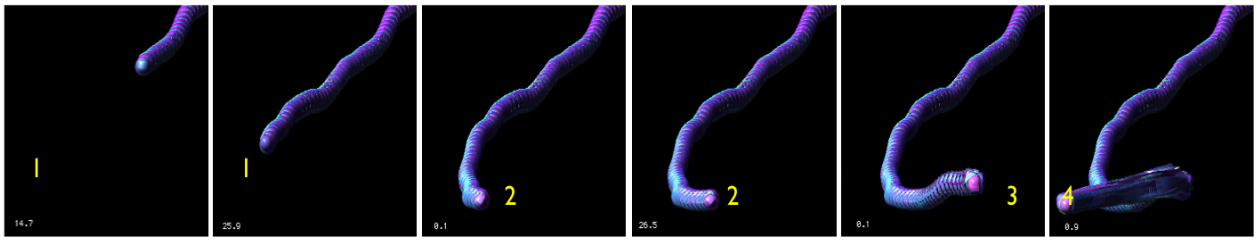


Figure 3: A tight group of swarm agents follows four sequential target points.

during the simulation through a mouse interface. Alternatively, target points can be overwritten or newly created through a video camera interface, where areas on the video canvas, for which movement is detected, act as attracting forces. So waving of your hand in front of the camera can navigate the swarms to different locations.[1]Figure 5 illustrates how the patterns of the *SwarmPainter* unfold over time.

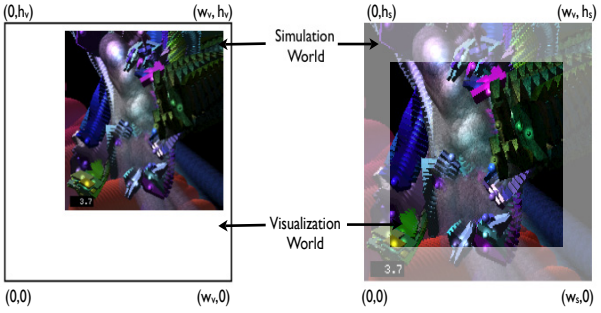


Figure 4: Lenses: Simulation world and visualization windows. The sizes of these 'lenses' are user-defined parameters. w_v and h_v are the width and height of the visualization world. w_s and h_s are the width and height of the simulation world.

[1]This is an updated version of the video-based control interface used in [5].

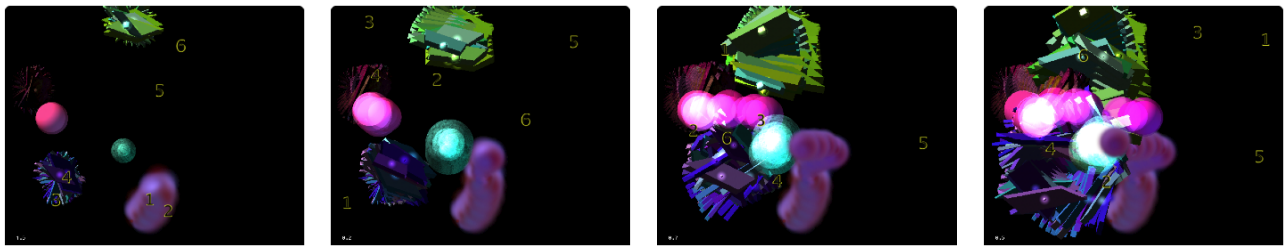
3.1.2 The SwarmPainter Lens

We call the portion of the swarm drawings visible to the user the *visualization world* (Figure 4). The simulation world is the space in which the three types of painter agents actually move. By changing the size of the visualization versus the simulation space, one can generate different artistic effects. A smaller visualization window focusses attention on a specific local area within the simulation world. A larger visualization window shifts the perspective, to the point that a set of images can be superimposed or particular background frames can be incorporated into the swarm painting.

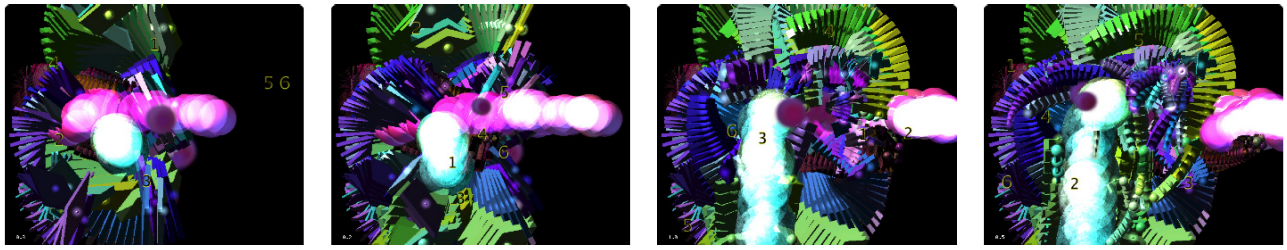
The widths and heights as well as the relative locations for both the simulation and visualization worlds are parameters within the system configuration file (and are also accessible by the evolutionary component of *SwarmPainter*).

3.1.3 Rendering

The *SwarmPainter* application (Figure 2) is created using the Cocoa and Quartz Composer frameworks under Mac OS X (10.4). All rendering is accomplished through Quartz Composer. The parameters for the rendering are set by the user via the parameter dialog or through a configuration file. Quartz Composer allows the user to work with high-level modules (patches) which are sub-routines for timers, user input, image filters and other video processing routines. The composition of patches provides a modular and flexible access point to sophisticated rendering routines, which control the actual visualization. Through the dialog window (Figure 6), all swarm behavior and appearance parameters



(a) t = 5 (b) t = 10 (c) t = 15 (d) t = 20



(e) t = 30 (f) t = 40 (g) t = 50 (h) t = 60

Figure 5: A typical run of a *SwarmPainter* simulation unfolding overtime. The numbers 1 to 6 in each rendered image represent the target points that direct the rendering agents. 60 time steps represent approximately 16 seconds on a MacBook Pro (2.16GHz Intel Core 2 Duo, 1 GB RAM).

can be controlled. Both of these interfaces therefore provide a high-level of intuitive control and accessibility to the artist/designer, who usually does not want to dive into the core routines of the swarm kernel.

The inputs to the patch are the set of 96 control parameters as outlined in Table 1. The patch then executes the input and renders a SwarmScape image as the output.

3.2 evoSwarmPainter

If we compare the computer's capability to generate numbers of images as opposed to manually generating these images by hand, the computer is vastly superior. However, this superiority does not necessarily apply towards the aesthetics: the quality of the image must still be assessed by the designer/artist. Setting target points for the swarms to migrate towards is very time-consuming, because *SwarmPainter* requires the user to manually set the target points through a cursor that guides the swarm along a path to create visual designs. In total, the simulation has 96 parameters (Figure 6) that give users control over the swarm interactions resulting in artistic renderings. Of course, setting these parameters manually becomes very tedious. Therefore we have wrapped the *Inspirica* [7, 8] evolutionary system around *SwarmPainter*.

The interactive evaluator in *Inspirica* allows users to steer the evolution by manually assigning a fitness value to each individual, starting a new iteration, and stopping the experiment (Figure 7). In each experiment, a collection of 'phenotype windows' is presented to the user. Each of the animated renderings shows the agents interacting for a particular set of control parameters generated by the evolutionary algorithm. The breeder assigns a fitness value between 0 and 10 to each of the simulations. A fitness of 0 will remove the individual from the population and thereby reduce the size of the population.

When an evolution experiment first starts up, *Inspirica* generates the initial population stochastically, i.e., the genotype in Equation 1 has randomly assigned values for the ninety-six parameters. The maximum and minimum values for the swarm parameters are described in Table 1 and are specified in a configuration file that *SwarmPainter* needs at the beginning of each experiment. For our experiments we use three swarm agents, two particle agents, and one texture agent. A parameter vector ($ctrl_A(t)$) for the six types of agents at time t is thus represented by:

$$ctrl_A(t) = (S_1(t),\ S_2(t),\ S_3(t),\ P_1(t),\ P_2(t),\ T_1(t)). \quad (1)$$

Applying genetic operators (mutation and crossover) to selected members of the current iteration then produces a new set of individuals for the next generation. Each operator has an assigned weight which determines its probability of application (Table 1). These weights are automatically adjusted, depending on an operator's contribution to a population's fitness gain (for details consult [4]).

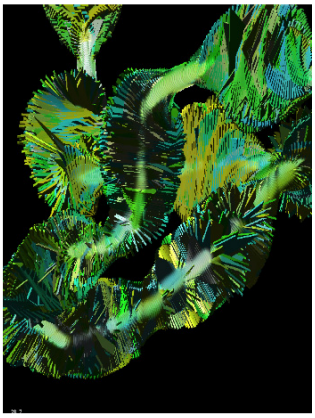
Mutation occurs by choosing a parameter x_k, $k = \{1, ...96\}$, from the list of parameters with fitness proportionate selection. A random value δ, following a uniform distribution within a user-defined range, is added or subtracted to x_k to produce a new parameter value:

$$x_k^{new} = min(x_k^{\max}, max(x_k^{\min}, x_k \pm \delta)), \quad (2)$$

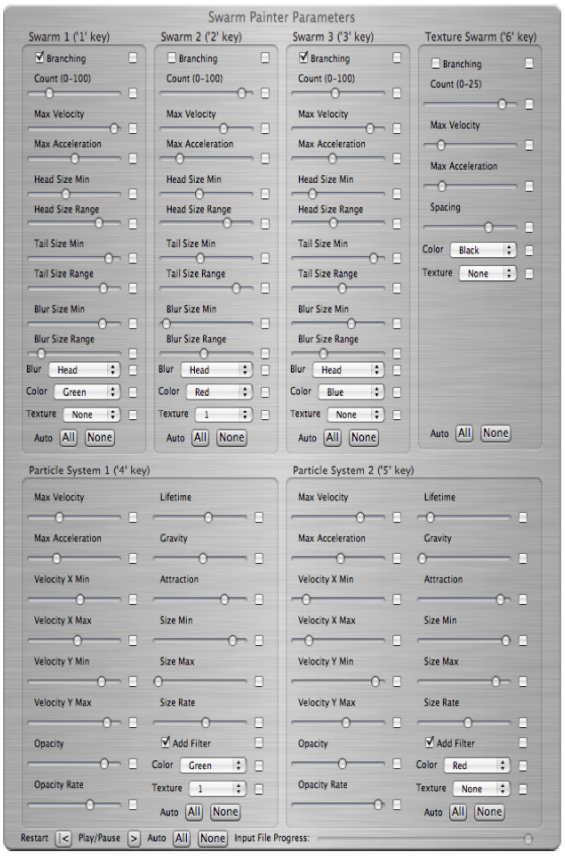
$$ctrl_A(t+1) = (x_1, x_2, \ldots, x_k^{new}, \ldots, x_{96}). \quad (3)$$

For recombination, we choose standard one-point crossover among two individuals. The two corresponding sections are interchanged at a randomly selected crossover point $i \in \{1, ..., 96\}$. For the experiments reported here, mutation and crossover are the only operators that we used as the resulting output turned out to be intriguing enough from an artistic perspective.

We observe the swarming behavior in each window for a set number of simulation time steps during which a SwarmScape unfolds through the interactions of the swarm, particle, and texture agents. The breeder/artist ranks each

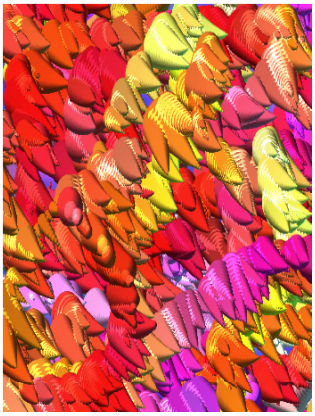


(b) Swarm agents

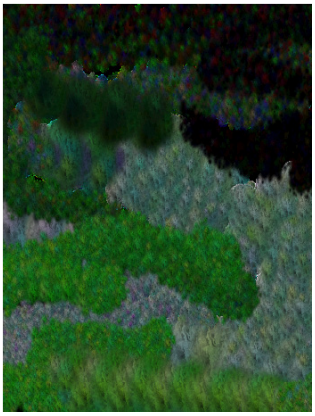


(a) Parameter dialog

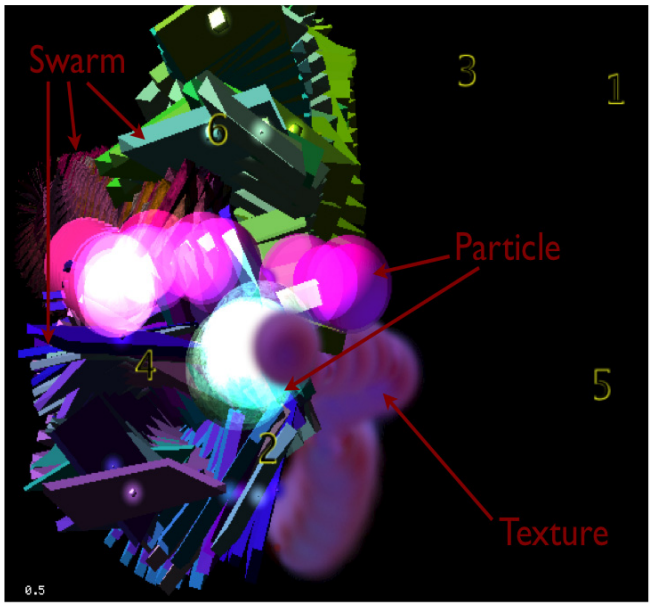
(c) Swarm agents



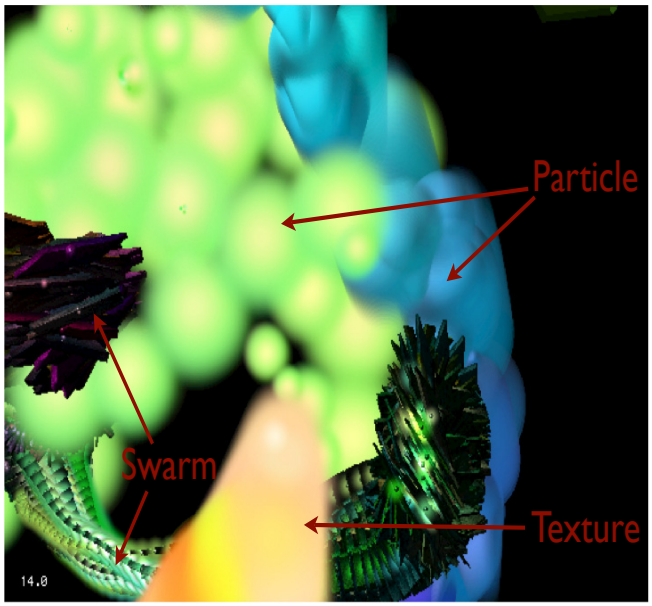
(d) Particle agents



(e) Texture agents

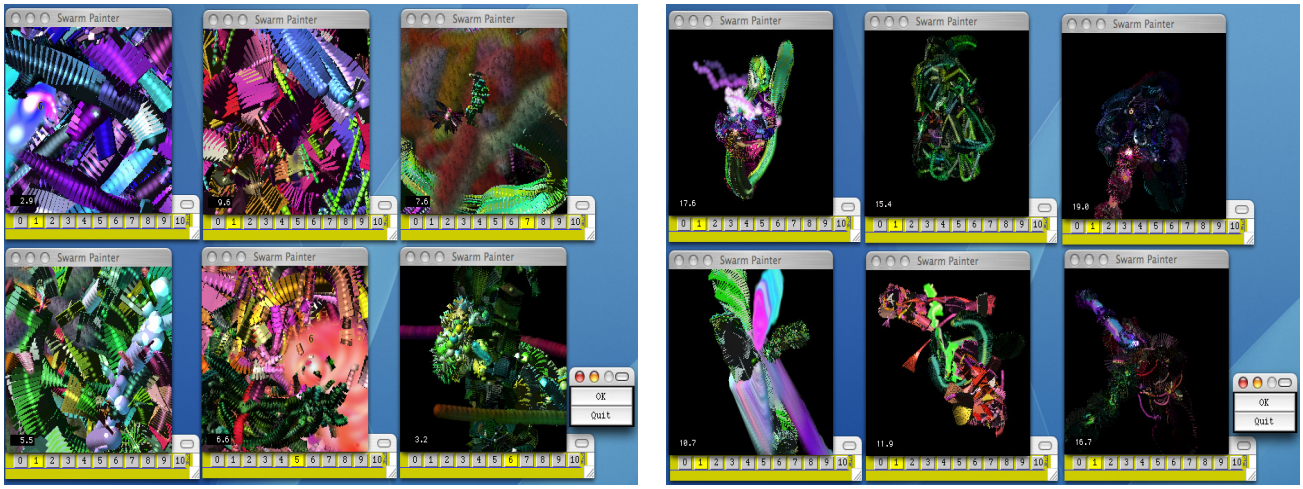


(f) Unfolding SwarmScape with target points



(g) Particles, swarms, and texture agents interacting

Figure 6: Control and appearance of *SwarmPainter* agents. (a) A dialog window gives access to a wide range of control parameters. (b) and (c) on the top exhibit the swarm agents' behavior for different parameter settings. (d) depicts the behavior of the particle agent. (e) shows the behavior of the texture agent for a specific set of parameters. Images (f) and (g) further illustrate the movements of the three agents (swarm, particle, and texture) together on the two-dimensional canvas.



(a) (b)

Figure 7: Interactive Evolution Interface: (a) A screen shot of the interactive interface of *Inspirica* is displayed here. The main control panel is on the right that allows the user to create the next set of *SwarmPainter* simulations from the current iteration ("OK") or to terminate the breeding ("Quit"). The user can assign a fitness value to the solutions through the individual evaluation panels. (b) For this example, agent parameter values (Table 1) are more restricted and result in more compact SwarmScapes.

SwarmScape, based on either how closely they generate a desired pattern or how interesting the newly discovered patterns look. Based on these interactively assigned fitness values, a new generation of individuals is created by applying mutation and crossover to the parameter vector of the current iteration (Figure 8). Because the evolution is guided by a human evaluator, the artist might give scores to designs in a highly inconsistent fashion as he/she changes his/her mind about desirable features during evolution. Therefore, the continuous creation of new and surprising forms based on the fittest from the previous iteration is important.

4. DISCOVERY OF DESIGN PATTERNS: THE ARTIST'S VIEW

For the artist, evolutionary computer design offers the potential for unique stylistic direction through huge numbers of design iterations that are impossible to generate via conventional computing or manual drawing. This iterative process establishes a symbiotic artist-machine partnership that results in unprecedented artistic expressions.

Because the computer does not stop, and continually modifies the visuals, the role of the artist becomes one of a voyeur who massages the image as opposed to direct image manufacture. Initially the artist submits an image or form into the matrix and then accepts the role of an artistic partner by cooperating with the computer. The process of watching six related images transforming concurrently is the real life equivalent to watching several different television programs at one time on fast-forward. As a consequence, the evolutionary process forces the artist into a fast-forward creative mode: this is just to keep up and make good judgments on the weighting of the transformations. Like a conversation of six individuals in a think tank, the six fitness evaluations suggest new connections for design departure points beyond what is immediately perceived.

In order to grasp the rapidly changing images, appreciation of the components and their relative complexities, engendered in art vocabularies, must be reduced. In the rudimentary stages of the evolutionary design, evaluation is reduced to form, space, rhythm, and color. If decorative embellishments like texture, materials, and surface are not stripped away, it becomes difficult to filter through the complexity and comparatively quantify each unique iteration of the fitness function screens. The machine actually forces a reductive "less-is-more appearance" advocated in architecture and Bauhaus artists like Joseph Albers.

Qualitative ranking within each fitness function (0-10) trains the system to evolve in unique image directions that are influenced by the artist's design bias. A design favoritism specific to the sensibility of the individual artist personalizes the image or form. When the artist recognizes a need for a design shift because of undesirable mutation, she/he alters the fitness functions. However, the result is a gradual shift through future iterations as opposed to instantly changing the flow of the evolution. This reciprocal tolerance and the requirement for compromise is a true human-machine partnership: both machine and user must wait and accept each other's operations.

Examples of the outcome of this interplay between machine and artist—tool and swarm sculptor—are illustrated in Figure 9. All of these images have also been re-rendered in high resolution (at 300 DPI) and realized as large (6 feet $\times$ 8 feet) plots on canvas.

5. CONCLUSION AND FUTURE WORK

We demonstrated *SwarmPainter* (Figure 9) as an example of an evolutionary design tool that utilizes interactions of swarming agents to dynamically generate form and structure on a two-dimensional canvas. We show that the artist remains a crucial partner within the overall endeavor of creating artistic artwork in cooperation with the evolutionary design generator.

The primary weakness of interactive evolutionary inter-

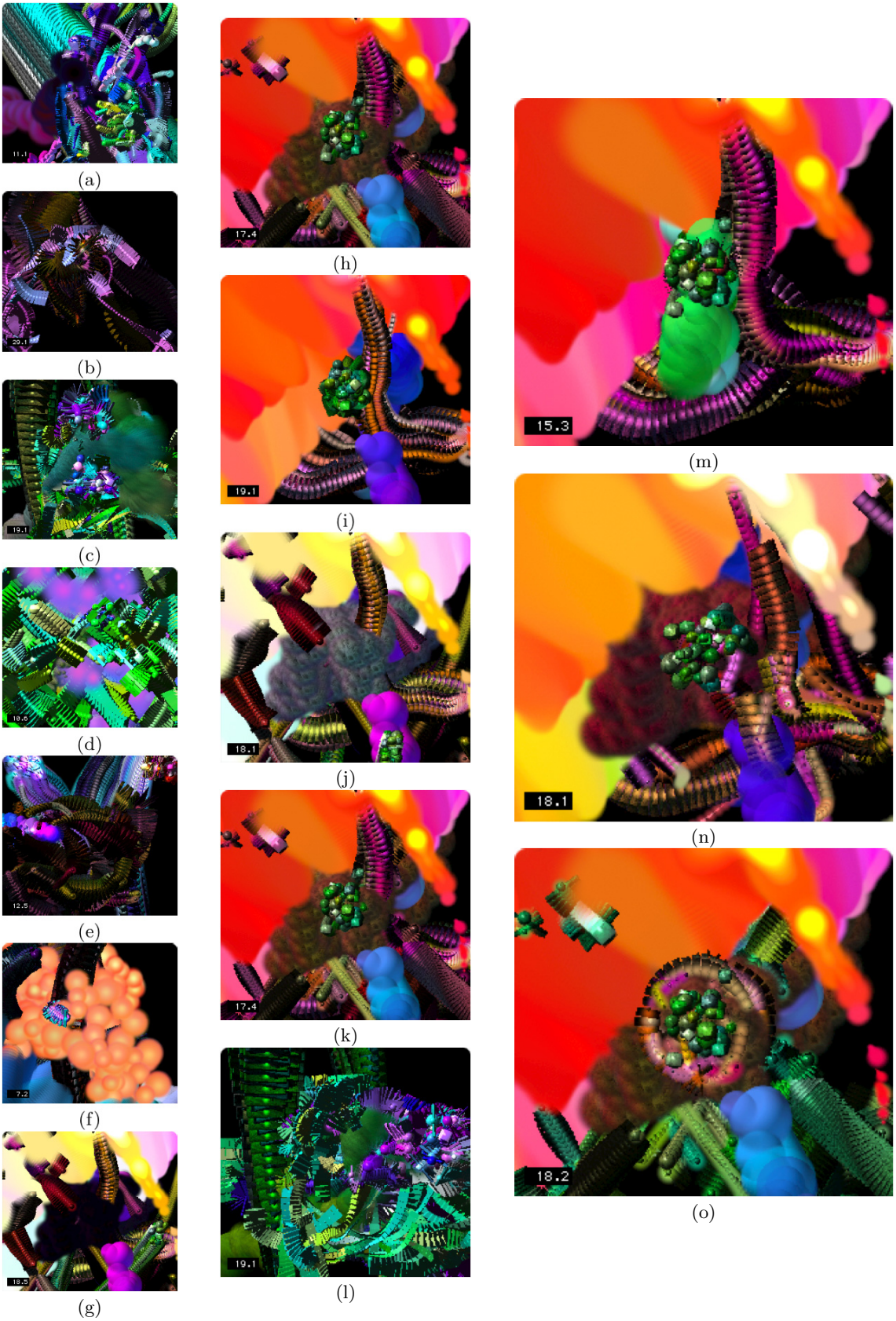


Figure 8: A typical run of the *SwarmPainter* simulation. The user is presented with seven phenotype windows at iteration 1 (column 1). Giving an individual a fitness of 0 deletes if from the gene pool. For this particular experiment, we had five individuals in iteration 7 (column 2) and only three individuals remaining by iteration 14 (column 3). Each of the SwarmScapes was generated over 240 time steps.

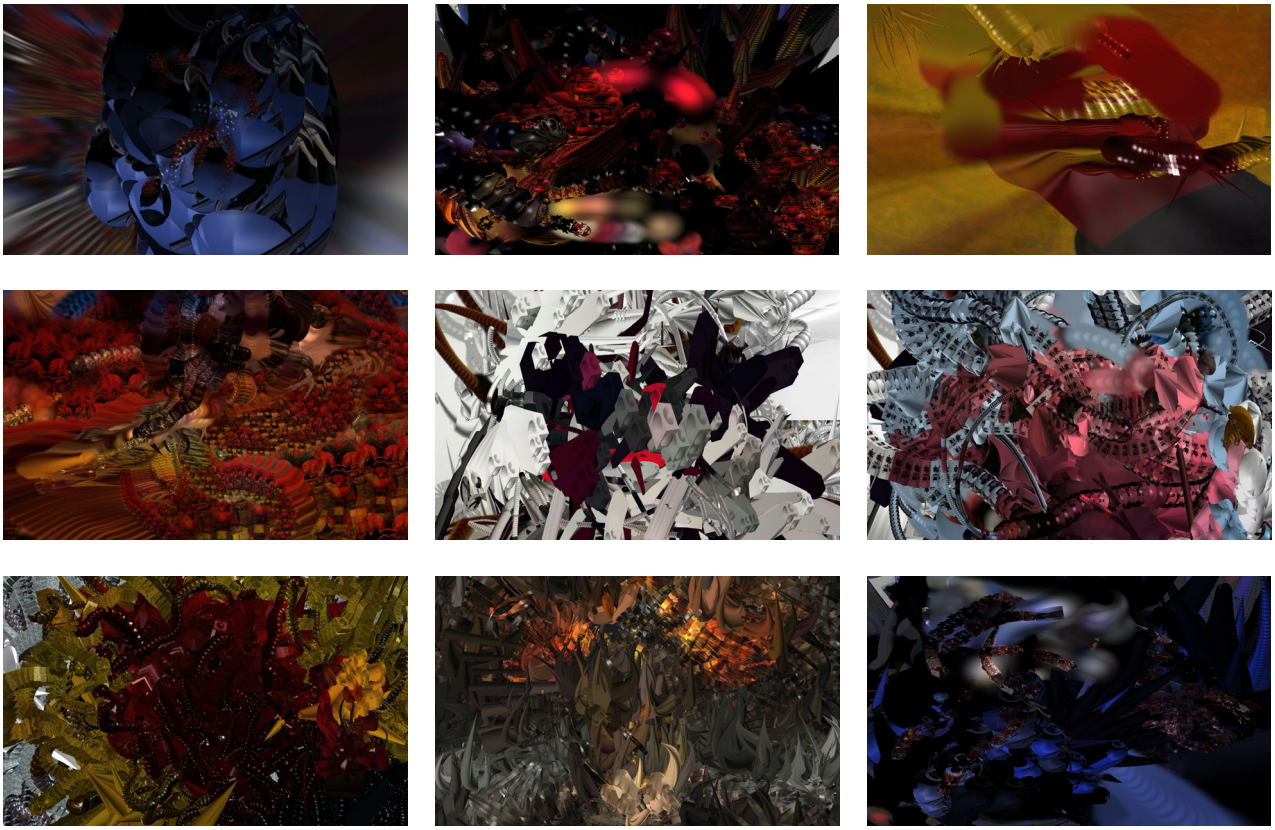


Figure 9: Images generated with *SwarmPainter*.

faces is that the users become quickly exhausted while evaluating screen after screen of populations. This is a general concern with IEC and has been noted in many places before (e.g., [9]). Therefore, we are beginning to examine mathematical models of aesthetics and other design and engineering criteria to incorporate those into *SwarmPainter*, thereby (partially) eliminating user interaction, and incorporating specific design constraints, such as symmetry, color correspondence, foreground-background relations, etc.

Animations of the results described in this paper, along with a gallery of images created with *SwarmPainter* can be found at our website: http://www.swarm-design.org.

6. REFERENCES

[1] P. Bentley and D. Corne. *Creative Evolutionary Systems*. Morgan Kaufmann, 2002.

[2] P. J. Bentley. *Evolutionary Design by Computers*. Morgan Kaufmann, San Francisco, CA, USA, 1999.

[3] R. Dawkins. *The Blind Watchmaker*. W.W. Norton and Company, New York, London, 1996.

[4] C. Jacob. *Illustrating Evolutionary Computation with Mathematica*. Morgan Kaufmann, San Francisco, CA, USA, 2001.

[5] C. Jacob, G. Hushlak, J. Boyd, M. Sayles, P. Nuytten, and M. Pilat. Swarmart: Interactive art from swarm intelligence. *Leonardo*, 40(3), 2007.

[6] C. Jacob and S. von Mammen. Swarm grammars: growing dynamic structures in 3d agent spaces. *Digital Creativity*, 18(1):54–64, 2007.

[7] H. Kwong. Evolutionary design of implicit surfaces and swarm dynamics. Master's thesis, University of Calgary, Calgary, AB, Canada, 2003.

[8] H. Kwong and C.Jacob. Evolutionary exploration of dynamic swarm behaviour. *IEEE Congress on Evolutionary Computation*, 2003.

[9] P. Machado and J. Romero, editors. *The Art of Artificial Evolution*. Natural Computing Series. Springer, 2007.

[10] S. Rooke. Eons of genetically evolved algorithmic images. In P. Bentley and D. Corne, editors, *Creative Evolutionary Systems*. Morgan Kaufmann, 2002.

[11] K. Sims. Artificial evolution for computer graphics. In *SIGGRAPH '91: Proceedings of the 18th annual conference on Computer Graphics and Interactive Techniques*, New York, NY, USA, 1991. ACM Press.

[12] H. Takagi. Interactive evolutionary computation: Fusion of the capabilities of ec optimization and human evaluation. *Proceedings of the IEEE*, 89(9):1275–1296, 2001.

[13] S. Todd and W. Latham. *Evolutionary Art and Computers*. Academic Press, Orlando, FL, USA, 1994.

[14] T. Unemi. Sbart 2.4: Breeding 2d cg images and movies and creating a type. *Knowledge-Based Intelligent Information Engineering Systems*, 1999.

[15] S. von Mammen and C. Jacob. Evolutionary swarm design of architectural idea models. In *GECCO 2008: Genetic and Evolutionary Computation Conference*, Atlanta, Georgia, USA, 12-16 July 2008.