

Aesthetic Video Filter Evolution in an Interactive Real-time Framework

Matthew Lewis

The Advanced Computing Center for the Arts and Design (ACCAD)
The Ohio State University, Columbus OH 43212, USA

lewis.239@osu.edu,

WWW home page: <http://www.accad.ohio-state.edu/~mlewis/>

Abstract. A data-flow network-based interactive evolutionary design framework is presented which will provide a testbed for the development and exploration of a diverse range of visual artistic design spaces. The domain of real-time layered video filters is focused on as the primary example. The system supports both real-time video streams and pre-recorded video. Issues of stylistic signature, GA vs. GP-based approaches, rapid tuning of fitness distributions, and desirable traits of generic evolutionary design systems are discussed.

1 Introduction

Artists and designers increasingly rely on an incredible array of complicated software which often takes years of study to learn. Simultaneously, the explosion of the Web, blogs, customizable online games, and social networking software have created an environment in which individuals who are not professional artists or designers are creating vast amounts of digital content to the best of their abilities with the authoring software that is available to them.

While these early days of consumer generated digital design have been focused primarily on the generation and manipulation of text and images, the creation of expressive 3D graphics, interactive experiences, and manipulated streaming video are feasible in terms of cost, bandwidth, and processing power. Accessibility remains elusive however due to the complexity of interfaces, concepts, and representations.

The ideal digital content authoring interface for most people would behave with the omniscience of science-fiction: “Computer? Create an Irish pub. No, more 19th-century. Yes, that’s more like it.” While such intelligent software is problematic for numerous reasons, interactive evolutionary design (IED) systems attempt just this approach (“I like those, show me more like them. . .”) albeit in extremely limited domains. This paper will describe a new framework being developed to enable rapid prototyping of IED-based digital content design interfaces and representations. After the primary system is described, an initial example design space of real-time digital video filter evolution will be discussed in depth.

What are the needs of a system such as the one outlined above? If the system is to be successfully used has a testbed for experimentation in different domains requiring different approaches, then representational flexibility and system modularity are both key. We would like to be able to easily modify the interface as appropriate for different problem spaces.

Perhaps just as important in an academic environment, we would like to enable non-programmer meta designers to quickly develop solution spaces in novel domains. Finally, a development environment in which the predominant visual design qualities defining a solution space can be quickly and frequently tuned and (re)evaluated without resorting to laborious and fatiguing chores like recompilation is an absolute necessity.

While it has been common over the past decade for individual researchers to program IED systems for specific problem domain, it is rare to find more generic systems for visual design space representation. One of the likely factors is the substantial task of authoring a wide range of graphics primitives and corresponding manipulation routines.

The last few years have seen a monumental drop in the cost of digital content development software that is readily extensible via integrated scripting languages, SDKs, and user-friendly data-flow network programming interfaces. While such software a few years ago was roughly the same price as an automobile, such features are now accessible commercially for hundreds of dollars, or even for free in open source software.

By creating IED frameworks within the context of extensible digital content development software, the processing, representation, communication, and interface capabilities of such systems can be harnessed. This in turn can provide easy access to the authoring power of the software, without requiring users to completely comprehend the intricacies of the systems' interfaces, processing concepts, and representations. The creation of design spaces within this context provides a valuable educational opportunity for those learning about a specific problem domain. Finally, advanced designers can readily craft, explore, and refine massive solution spaces, using them generatively to rapidly produce large numbers of alternative designs.

2 Background

The evolutionary design systems used for creating and manipulating images are far too numerous to survey here (see [1][2][3].) The majority of image generation and animation systems (e.g., [4][5][6][7]) are variants on the techniques pioneered by Sims [8]. Most such software, in addition to producing images and animations, supports still image manipulation as well. In general however, these systems do not operate in real-time on streams of images and most often the generate-and-select cycle is tuned to maximize the resolution possible in the time that the user is willing to wait for a new population.

There have been few generic systems for visual IED. Rowley's software combined a toolkit for displaying and judging image based individuals, with a mod-

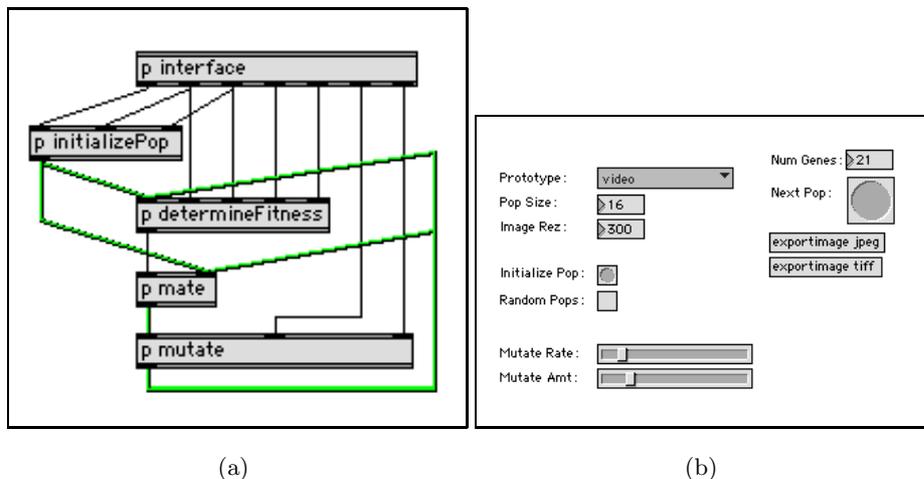


Fig. 1. The primary system patch and the evolution controls

ular system for mutating and crossbreeding expression based representations such as fractal images [9]. Pontecorvo was developing software primarily for consumer product design [10][11]. Lewis's generic system used data flow networks for solution spaces and genetic algorithm representations [12].

Perhaps most interestingly, Todd and Latham created a generic evolutionary design system called *PC Mutator* capable of interfacing with external existing Windows-based design software such as paint programs. Once parametric models were built (e.g., for designing cartoon faces), *PC Mutator* sent commands to external applications to create populations. The external application generated images of the individuals, which were then passed back to *PC Mutator* for display. Subjective fitness determination, mutation and crossover take place in *PC Mutator* [13][14].

3 Interactive Evolution Framework

Max and MSP have for years been among the cornerstones of computer music performance and composition software [15][16]. A little over a year ago, Max was formally extended to fully support a range of visual domains in an extremely well integrated fashion, via a set of objects called Jitter [15]. Jitter is at its core an extensible library for matrix manipulation. The matrix data in question might include images, video, sound, sensors, text, or anything which can be mapped into an N-dimensional numerical form.

The Max and Jitter environment is not a turn-key digital content authoring environment, but rather can be viewed as a toolkit for constructing such systems. Applications constructed within the Max/Jitter framework, once refined can be

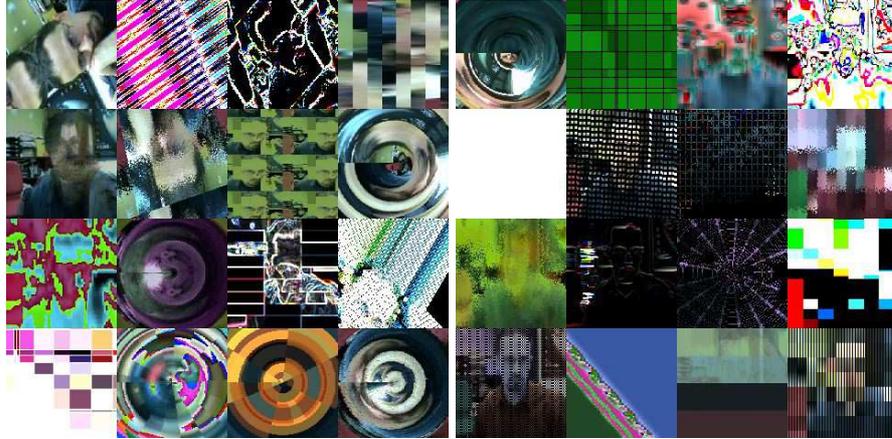


Fig. 2. Initial populations

packaged and distributed as standalone software. While the user can certainly extend the system via an SDK, the primary development approach is via a data-flow network programming paradigm. Figure 1(a) shows the top level network or “patch” (also “patcher”) for the system described below. With information flowing downward, the data path from the interface and the initial population, into an iterative cycle of fitness determination, mating, and mutation, can clearly be viewed. The encapsulation of details in sub-patches makes it easy for those learning about the system to quickly understand the architecture of the system, and eventually extend and modify the process. Furthermore, the system can be modified on-the-fly, with no stopping/compiling/linking/etc. The primary evolution interface can be seen in 1(b) showing the a typical range of controls for things such as mutation rate and amount, population initialization, and so on.

Design Domains Designs produced using a given IED system invariably show a strong *signature* [17]. Signature, in this context, refers to the lack of visual diversity and generality in the designs produced (e.g., the polar coordinate remapping appearing in many of the individuals in the left initial population in figure 2.) Most work produced using a given IED implementation shares extremely strong visual characteristics, identifying the work as having been produced by that system, regardless of user. Ideally sufficiently general design spaces containing all possible desirable solutions could be created, with a broad distribution of high fitness regions to make satisfactory convergence certain. However, this is much easier said than done.

Solution space authors must constantly weigh convergence speed and fitness against generality and signature. While a given representation might contain an image of the Mona Lisa [18], it’s extremely unlikely that a user of the system using a low-level representation (e.g., a simple math expression hierarchy) will

discover it in any reasonable amount of time interactively. As will be discussed below, basing our system on an easily extensible genetic algorithms approach allows for precise remapping of specific genes into “more desirable” value distributions for phenotype parameters.

Representation The framework used in this system relies on a fixed length vector of genes representing normalized parameter values. This is to be contrasted with the genetic programming approach involving hierarchical graphs of variables, constants, and functions, more typically used for evolving imagery. Several authors have commented on the comparative difficulty of controlling GP-based systems interactively. Margaret Boden claims Sims’ system “can not be used to explore or refine an image space in a systematic way [19].” Todd and Latham question whether structure-based mating and mutation “...gives enough control for artistic applications [20].” While the space of possibilities is generally more vast, practical convergence remains a serious issue. Musgrave puts it best when he says that GP-based systems “tend to be simultaneously more chaotic, hard to control, and creative [21].”

Architecture Choosing a software development environment such as Max and Jitter for infrastructure support means a framework capable of supporting real-time 3D graphics, networking, interaction, image creation, particle systems, lighting, and of course, video. Since video processing is one of Jitter’s primary strengths, it was an appropriate initial design domain to address. (The significant demand recently from the students and faculty in our departments of theater, dance, and fine art for real-time, data driven video manipulation capabilities for performance and installation has provided additional incentive.)

The initial GA framework was developed and tested using a very simple solution space for evolving color palettes, nearly identical to the one shown in [12]. Most IED systems which evolve populations of static or unchanging individuals in this manner can render and display each phenotype, requiring only that the genotype and image (or polygons, animation, etc.) be stored for each population. This “generate and forget” paradigm is impractical in domains requiring each individual to remain active during fitness evaluation, such as in the video filtering domain described below.

Once the user selects the fittest individuals (by clicking on them), they form the mating pool. Random pairings are then mated to form a new population of offspring. Though uniform crossover was primarily used, and only the selected individuals are mated, the modularity of the patches make it trivial to switch to single-point crossover, allow selected individuals to survive with some percentage chance, etc. These sort of modifications can even be made on-the-fly in the running system.

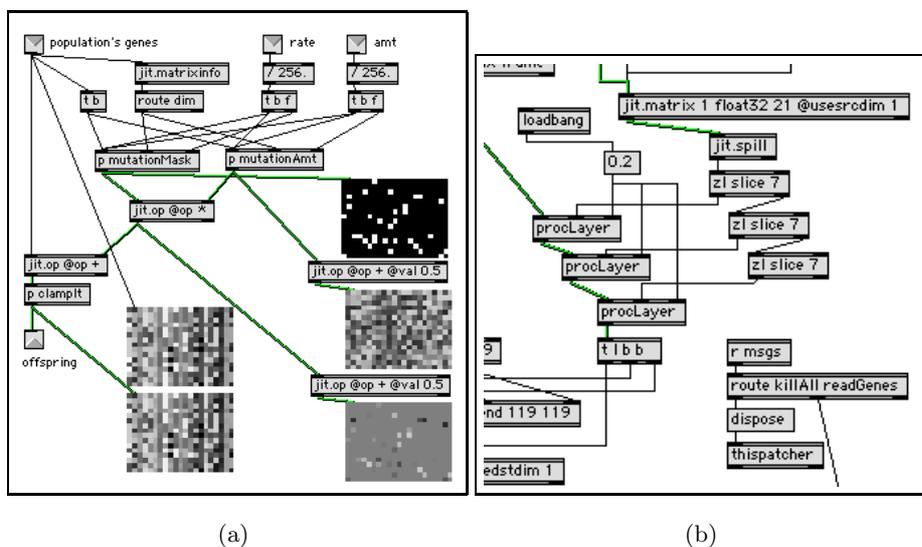


Fig. 3. The mutation patch on the left illustrates the mutation process. In the patch excerpt on the right, the matrix in the upper right holds the 21 genes for one individual. These are sliced off seven at a time for use by each of three video processing layers.

4 Video Filter Space

For the real-time video processing design space, a flexible “processing layer” approach was used. A single processing layer consists of a bank of different possible video filters. Nineteen were used for the work discussed in this paper. Each filter has one or more continuous parameters (three on average) and many have several different modes, each of which yields different visual results. Some examples of classes of video filters include adjustments to brightness, contrast, saturation, and hue, various spacial remappings, resamplings, and transformations, the addition of different types of noise, stochastic pixel position transformations, edge detection, thresholding, and feedback. The hierarchical patch-based nature of the software makes it simple to extend this by dropping in any new processing filters as desired.

Phenotypes are produced by slicing off blocks of g genes (currently $g = 7$) and passing them to a sequence of processing layer objects (figure 3(b)). In each layer, the first gene of each block functions as a “choice” gene determining which filter each layer will use. The remaining $g - 1$ genes in each layer function as “activation” and “parameter” genes, determining which filter modes are used and with what intensity or setting, respectively. Three layers of filters (21 genes) were used for the images in this paper.

Since different filtering processes’ modes have different visual “strengths”, the different filter modes can be manually assigned individual activation thresholds.

For a filtering mode to activate, the value of its activation gene must exceed the assigned threshold value. This is one means by which individual formal visual qualities of the solution space can be controlled by the meta designer, increasing or decreasing specific stylistic signatures.

Any attempts to correlate Euclidean distance in the genotypic space with a continuous visual/perceptual difference requires careful remapping of normalized gene values into often nonlinear filter parameter ranges. Perlin's bias and gain functions can be used to intuitively push and pull normalized gene values towards or away from extremes [22]. Even a simple function remapping genes' lower and upper domains ($[0.0, 0.5]$ and $[0.5, 1.0]$) into $[low, default]$ and $[default, max]$ ranges respectively can be enormously useful, as well as simple to adjust on-the-fly as solution spaces are refined.

The use of the single filter-choice gene in each layer produces local minima since an individual filtering layer can only transition from its current filter to either neighboring filter. A simple extension to address this will be the use of two or three genes controlling filter choice. In a two or three dimensional filter space, a filter could mutate into one of its eight or twenty-six neighbors, respectively.

The mutation sub-patch in figure 3(a) shows the generation of the mutation mask (upper right) which is applied to random mutation values (middle right) to produce final gene mutations (lower right). In the matrix images within the patch, pixel rows represent individuals, and columns are individual genes. Interactive sliders allow the user to control the percentage of the population's genes that are mutated, as well as the amount they are mutated.

Note that a mutation mask should be used to control the visual step size in individual dimensions by scaling the applied mutation amplitude differently for selected genes, specifically in this case for the very sensitive filter selection gene within each of the processing layers.

Results The benefits of a real-time architecture become particularly evident when the user is suddenly able to probe the *behavior* of the individuals in a population via the interactive feedback loop. The "credit assignment" problem of determining what components were responsible for the fitness of an individual is often implicitly addressed in IED via experience, e.g., a user might discover they should breed individuals which appear to compose *mod* and *fbm* functions which often lead in interesting directions, but perhaps not choose the ones with attractive high frequency aliasing which will probably turn to mush in another generation, etc. In a real-time system however the user finds they can often use more information about the visual robustness of a given solution, instead of waiting until the next population is produced only to find their choice was sitting atop a tiny fitness peak.

Figure 4 shows a set of sample converging solution populations, as well as three frames of one population's behavior over time. The images shown were recorded with the system running on a 1ghz G4 laptop at about 6-10fps sampling a live video feed at 160x120 resolution. It is expected that a dual 2.0ghz G5 should allow faster frame rates, higher resolutions, and additional layers. Note

that while *live* video (from a FireWire camera) is being discussed, the system can just as easily display and process looping digital video clips, or arbitrary other DV or analog video sources (with the use of an inexpensive analog-to-DV converter) such as video tapes, live television, teleconferencing, etc.

5 Conclusion

As the approach begun here is extended, it is hoped that an extremely flexible system can be produced which will allow artists and designers (both professional and aspiring) to create expressive digital content using a simple interactive evolution paradigm. Usage of data-flow network authoring and real-time systems should aid in the rapid development of solution spaces by non-computer scientists.

In the near future we are looking forward to investigating the wide range of interactive data domains which this software allows one to represent including combinations of shapes, images, geometry, particles, lighting, materials, motion, environmental sensors, networking, text. . . we've only just begun to explore the potential spaces to which evolution-based techniques could be applied within this framework. In addition to expanding the range of domains, one of the next extensions will be to explore interface options to make the fitness evaluation resolution more interactively dynamic using OpenGL-based zoomable user interface (ZUI) approaches.

Within the video manipulation domain, currently only sequential processing filters have been used. Additional processing power should allow for the addition of compositing operators to parametrically combine two video filter sequences in different ways. Also a library of interchangeable patches giving different options for mating and fitness evaluation needs to be developed. There are many options for extracting and extending the genetic representation so that new domains can be more rapidly created from older ones via shared sub-patches. Ideally, the transition between different filters would be more more continuous, with the center of a filter's region of genetic space mapping to a full activation of the effect which then fades in strength as it transitions into regions of neighboring filters (within one filter layer.)

Educationally, this system will serve as a tool for our art and design students to explore spaces of design options (literally and metaphorically), as well as encourage them to seriously consider *how* they explore these spaces of possibilities, by allowing them to explicitly attempt to design and refine them.

The promise of interactive evolutionary design is the creation of software based on the encouragement of exploration of many ways of seeing, as well as the critical examination of current and future possibilities. The merging of art, design, and technology in this new paradigm in order to represent, study, and harness diversity interestingly mirrors the corresponding need for its increased appreciation in the larger world.

References

1. Bentley, P.J.: *Evolutionary Design by Computers*. Morgan Kaufmann (1999)
2. Lewis, M.: Visual aesthetic evolutionary design links. <http://www.accad.ohio-state.edu/~mlewis/aed.html> (2003)
3. Takagi, H.: Interactive Evolutionary Computation: Fusion of the Capabilities of EC Optimization and Human Evaluation. *Proceedings of the IEEE* **89** (2001) 1275–1296
4. Greenfield, G.: Mathematical building blocks for evolving expressions. In: *BRIDGES: Mathematical Connections in Art, Music and Science* July 28–31. (2000)
5. Musgrave, F.K.: Genetic programming, genetic art. <http://www.wizardnet.com/musgrave/mutatis.html> (2003)
6. Rooke, S.: The Evolutionary Art of Steven Rooke. <http://www.azstarnet.com/~srooke/> (2001)
7. Unemi, T.: SBART2.4: Breeding 2D CG Images and Movies, and Creating a type of Collage. In: *The Third International Conference on Knowledge-based Intelligent Information Engineering Systems*, Adelaide, Australia, August, Geneva, Switzerland (1999) 288–291
8. Sims, K.: Artificial evolution for computer graphics. *ACM Computer Graphics* **25** (1991) 319–328
9. Rowley, T.: A toolkit for visual genetic programming. Technical Report GCG-74, The Geometry Center, University of Minnesota (1994)
10. Emergent Design: Emergent Design. <http://www.emergent-design.com> (2000)
11. Pontecorvo, M.S.: Designing the undesigned: Emergence as a tool for design. In: *Proceedings of Generative Art 1998*, Milan, Italy. (1998)
12. Lewis, M.: Aesthetic evolutionary design with data flow networks. In: *Proceedings of Generative Art 2000*, Milan, Italy. (2000)
13. Todd, S.: Personal communication (2000)
14. Todd, S., Latham, W.: The mutation and growth of art by computers. In Bentley, P.J., ed.: *Evolutionary Design by Computers*. Morgan Kaufmann (1999) 221–250
15. Cycling '74.: Max/MSP and Jitter software. <http://www.cycling74.com> (2003)
16. Winkler, T.: *Composing Interactive Music: Techniques and Ideas Using Max*. MIT Press, Cambridge, MA (1998)
17. Rowbottom, A.: Evolutionary art and form. In Bentley, P.J., ed.: *Evolutionary Design by Computers*. Morgan Kaufmann (1999) 261–277
18. Whitelaw, M.: Breeding aesthetic objects: Art and artificial evolution. In Bentley, P.J., Corne, D., eds.: *Proceedings of the AISB'99 Symposium on Creative Evolutionary Systems (CES)*. Morgan Kaufmann (1999)
19. Boden, M.A.: Agents and creativity. *Communications of the ACM* **37** (1994) 117–121
20. Todd, S., Latham, W.: *Evolutionary Art and Computers*. Academic Press (1992)
21. Musgrave, F.K.: Genetic textures. In Ebert, D., ed.: *Texturing and Modeling: a Procedural Approach*. Academic Press (1998) 373–384
22. Perlin, K.: Noise, hypertexture, antialiasing, and gestures. In Ebert, D., ed.: *Texturing and Modeling: a Procedural Approach*. Academic Press (1998) 209–274

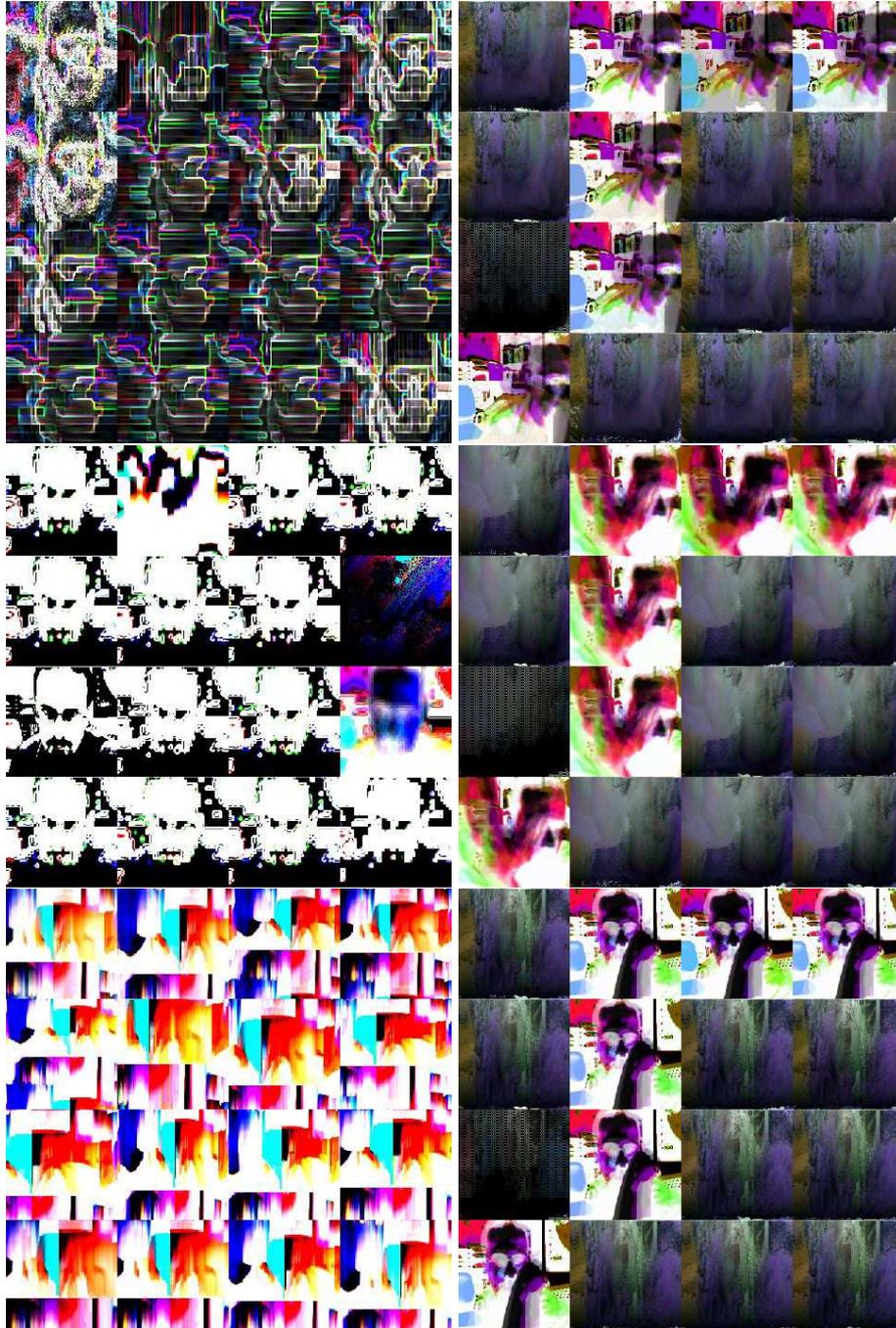


Fig. 4. The left column shows several converging populations, while the right column shows several frames of the same population's behavior over time.