Artificial Art: Image Generation using Evolutionary Algorithms*

Johannes Rückert

FB Informatik
FH Dortmund
Emil-Figge-Str. 42
44227 Dortmund
saviola@stud.fh-dortmund.de

Abstract: This paper gives an overview of the principles of evolutionary algorithms–and especially genetic programming–and how they are used in the context of evolutionary art. Additionally, various methods for image creation and evaluation are introduced. Genetic programming allows for the automatic solving of problems using a given set of functions, from which individuals are generated and then–after being evaluated using a given fitness function–optimized through mutation and recombination. The software packages *Reflection-based Genetic Programming* (ReGeP), which allows the function set to be decoupled from the framework or application in which it is used, and the *Java package for evolutionary art* (Jpea), which offers a small collection of reusable classes and an exemplary evolutionary art application, are shortly described.

Abstract: Dieses Paper gibt einen Überblick über die Prinzipien evolutionärer Algorithmen und besonders genetischer Programmierung, und wie diese im Kontext der evolutionären Kunst Verwendung finden. Zusätzlich werden eine Reihe von Techniken zur Bilderstellung und -bewertung vorgestellt. Genetische Programmierung ermöglicht die automatische Lösung von Problemen durch die Vorgabe eines Funktionssatzes, aus dem Individuen basierend auf Prinzipien der natürlichen Evolution erzeugt, bewertet, selektiert, mutiert und rekombiniert werden. Die beiden Softwarepakete *Reflection-based Genetic Programming* (ReGeP), welches die Entkopplung des Funktionssatzes von der Applikation oder dem Framework in dem er benutzt wird erlaubt, und das *Java package for evolutionary art* (Jpea), welches aus einer Sammlung wiederverwendbarer Klassen und einer beispielhaften Applikation zur Generierung evolutionärer Kunst besteht, werden kurz vorgestellt und beschrieben.

1 Introduction

Evolutionary algorithms is a generic term for nature-inspired optimization algorithms which generally use the concepts of populations, selection, mutation and recombination, and can be used for problems where solutions are hard to create manually and easy to evaluate.

*Project and Bachelor's thesis supervised by Prof. Dr. Friedrich and Prof. Dr. Scheben. This paper was published as [Rü16].

Evolutionary computation (EC) is a field that covers techniques simulating different aspects of evolution and was formed during the 1990s in an effort to bring together researchers from the fields of evolutionary programming, evolution strategies and genetic algorithms [BFM97].

Genetic programming (GP) is another more recent evolutionary algorithm, first formally described in [Koz92], in which executable computer programs are evolved. Evolutionary algorithms and GP are described in section 2. Evolutionary art (EA) is a field in which evolutionary algorithms are applied to evolve works of art. The field was inspired by [Daw86] and one of its pioneers was Sims, who used genetic programming to evolve LISP expressions, which were then used to generate images [Sim91]. EA, image creation, and image evaluation are expanded on in section 4.

This paper is structured as follows: Section 2 will introduce evolutionary algorithms in general, while section 3 will describe how reflection was used in the context of GP. After that, section 4 will describe the general concepts of EA, as well as techniques used for creating and evaluating images. The final sections 5 and 6 will give an overview of the results, including example images, and a short conclusion.

2 Evolutionary algorithms and evolutionary computation

Evolutionary computation is a field that covers techniques simulating different aspects of evolution. According to [BFM97, p. vii], "The field began in the late 1950s and early 1960s as the availability of digital computing permitted scientists and engineers to build and experiment with various models of evolutionary processes." However, it was formed only in the 1990s to bring together various subgroups of different EC paradigms which had emerged during the previous decades, most importantly evolutionary programming (EP), evolution strategies (ES) and genetic algorithms (GA) (see section 2.2). Also during the 1990s, [BFM97] was produced as the "first clear and cohesive description of the field" [BFM97, p. vii].

Evolutionary algorithms is a generic term for nature-inspired optimization techniques, and generally refers to the algoritms used in the field of EC.

What really connects the three EC paradigms of EP, ES, and GA, is the simulation of an evolutionary process (see section 2.1), i.e. "the reproduction, random variation, competition, and selection of contending individuals in a population" [BFM97, p. A1.1:1], to solve an optimization problem.

Reasons for the simulation of evolutionary processes come primarily from the fields of optimization, robust adaptation, machine intelligence and biology, and "The ultimate answer to the question 'Why simulate evolution?' lies in the lack of good alternatives." [BFM97, p. A1.1:2]

2.1 The algorithm

Evolution, as defined by the neo-Darwinian paradigm, is "the inevitable outcome of the interaction of four essential processes: reproduction, competition, mutation and selection" [BFM97, p. A2.1:1].

Reproduction, in terms of EC, happens by transferring an individual's genetic information to progeny, either asexually, just copying the individual, or sexually, by performing *recombination*. Genetic information is usually subject to *mutation*, which serves to (re)introduce greater genetic diversity. Competition is simulated by using a fixed population size and *selection* determines which individuals survive and/or reproduce.

Individuals are defined by their genetic program, or *genotype*. The genotype is translated into the *phenotype*, which is used to determine the *fitness* of individuals, which in turn is used in the selection process. The genotype-phenotype mapping is often subject to *pleiotropy* (i.e., a single gene influences several phenotypic traits) and *polygeny* (i.e., several genes influence a single phenotypic trait) so that there is no *strong causality* for changes in the genotype and in the phenotype [Bä96].

Selection is performed, as mentioned before, only on the phenotypes, while reproduction is performed only on the genotypes of individuals.

Evolution is an optimization process, and, as stated by the no-free-lunch theorem [WM97], it is more useful than other optimization processes for some problems, and less useful for others; it is, like other optimization processes, not a process leading to perfection (for most problems) [BFM97].

In the following, the most basic process shared by all evolutionary algorithms will be laid out. The algorithm consists of four steps [Atm94]:

Step 1: An initial population of predefined size is randomly created. Alternatively, the initial population may be *seeded* with specific individuals. Seeding can be used to continue an earlier evolutionary process or to accelerate the optimization process by providing individuals which are known to have a higher fitness score than the average randomly generated individual, or to ascertain a higher genetic diversity in the initial population than a completely random initialization offers.

Step 2: In this step, the population is replicated. Replication happens by applying one or more genetic operators on the existing population. Examples for such operators are mutation, recombination or reproduction.

Step 3: The individuals of the population are evaluated by a fitness function and selection is performed. Sometimes, selection is performed before replication and sometimes the genotype has to be translated into a phenotype. Through selection, the new population is determined. In some algorithms, new individuals are chosen from both parents and children, in others only the children are considered.

Step 4: Repetition. Steps 2 and 3 are repeated until one of potentially several termination criteria is met. Criteria might be a maximum number of iterations or the quality of the solution.

2.2 Techniques

This section will shortly explain the four techniques of evolution strategies, evolutionary programming, genetic algorithms, and genetic programming.

Evolution strategies were first developed in the 1960s by Bienert, Rechenberg and Schwefel. They were created to cope with the "impossibility to describe and solve [certain] optimization problems analytically or by using traditional methods" [Bä96, p. 67] by introducing random mutations that were meant to imitate mutations found in nature. The first ES was the so called $(1 + 1)$-ES (the first number indicating the number of individuals in the population and the second the number of children per individual), where only one individual exists which is mutated to create the offspring and then evaluated and compared to the parent individual–the better of the two is retained. Similarly, $(\mu + 1)$-ES (a variable number of individuals with one child each), and $(\mu + \lambda)$-ES (more than one child per individual) were introduced to enhance the original version. Additionally, (μ, λ) (here, the next generation of individuals is chosen only from the offspring) was developed [Sch75, Sch77, Sch81]. Evolution strategies, especially (μ, λ)-ES, are still widely used today.

Introduced by Fogel in 1964, **evolutionary programming** is based on finite-state-machines (FSMs) which are used to match an input sequence to a certain output sequence. The fitness of an individual is measured by the percentage of correct output values. The individuals (first a $(1 + 1)$-selection was used which was later extended to a $(\mu + \lambda)$-selection to avoid becoming stuck in local optima) were mutated slightly and selection was performed among parents and offspring. No recombination is used [Bä96, BFM97, BFM99].

Genetic algorithms were first formally introduced in [Hol75]. They differ from other evolutionary algorithms primarily in three ways: Their representation (bitstrings), their selection method (fitness proportionate selection) and their primary genetic operator (crossover). There have, however, been many variations of GA since, which have used different forms of selection and representations other than bitstrings, so that only the focus on crossover remains as uniquely identifying [BFM97, B1.2:1]. In its original form, as proposed by Holland, GA use bit strings of fixed length as representation. When evaluating individuals, segments of bits (usually of equal length) are interpreted as integers which are then used in the context of the problem which is to be solved. The algorithm works mostly as described in section 2.1. After evaluation, individuals are chosen for crossover probabilistically based on their fitness value. The idea behind this it to combine beneficial traits of parents by creating offspring who is basically a mix of both parents' genes. There are different kinds of crossover: the simple one-point crossover, which exchanges all bits to the right of the crossover point; and the more powerful multi-point crossover, which exchanges bits between two points–several segments can be formed and exchanged this way. Uniform crossover randomly decides for each bit from which parent it is taken. Mutation is mostly used as a background operator to reintroduce traits that were lost through convergence in the population [Bä96].

"**Genetic programming** is an evolutionary computation technique that automatically solves problems without requiring the user to know or specify the form or structure of the solution in advance. At the most abstract level, GP is a systematic, domain-independent method

for getting computers to solve problems automatically, starting from a high-level statement of what needs to be done" [PLM08, p. 1]. In GP, computer programs are evolved using the basic evolutionary concepts and processes of population initialization, selection and reproduction. It is very similar to GA regarding the basic algorithm: a random population is generated, individuals are evaluated, crossover (and, optionally, mutation) is performed and selection determines which individuals may reproduce. The fundamental difference between GP and GA lies in the representation: While GA have been implemented using many different representations (bit strings only being the first), most of these representations encode domain knowledge into the genotype by using a fixed size and / or structure. These limitations make exploring the theoretical foundations easier, but do not permit an application to different problems–for each problem a new representation has to be devised. John Koza's idea was that "for many problems in machine learning and artificial intelligence, the most natural representation for a solution is a computer program (i.e., a hierarchical composition of primitive functions and terminals) of indeterminate size and shape, as opposed to character strings whose size has been determined in advance." The initial–and still widely used–representation for such program structures were trees. Other representations have been used in Linear GP (linear lists of commands, [BNKF98]) and Cartesian GP (graphs, [Mil11]).

3 Reflection-based Genetic Programming

> "Reflection-based Genetic Programming (ReGeP) is an extension to frameworks supporting Genetic Programming which provides reflection-based creation of the GP function set, allowing for absolute decoupling of the functions from the GP framework."[1]

ReGeP was developed and documented primarily in [Rü13b]. As mentioned in the quote above, ReGeP is not a GP framework itself, but instead serves as an extension to GP frameworks. Possible choices of frameworks include JGAP[2] or ECJ[3], the former being the only one supported by ReGeP at the moment. The task of the GP framework is to provide an implementation of the general genetic algorithm, i.e., population initialization, selection, recombination and mutation (and possibly more). The GP system using the framework is required to provide a fitness function which evaluates the individuals, a GP function set and various configuration parameters. The usual workflow for creating a GP function set includes creating classes which implement a certain interface or extend a certain class as required by the framework. This means that switching the framework means creating (at least) new wrapper classes for the GP functions. The goal of ReGeP is to allow the function set to be written as Java classes and methods without any dependencies, so that they can be reused in other context or existing functions to be used in GP systems without wrapping or reimplementing them. To achieve this loose coupling, ReGeP uses *reflection*.

[1] `https://gitlab.com/regep/regep`, accessed November 28, 2015.
[2] `http://jgap.sourceforge.net/`, accessed November 28, 2015.
[3] `http://cs.gmu.edu/~eclab/projects/ecj/`, accessed November 28, 2015.

> "Reflection in an OO language is where a program is able to discover things about itself at run time, such as the class of an object, the fields associated with it, its superclass, the set of interfaces it implements, its set of constructors, and the set of methods that can be invoked on it" [Luc04].

Using reflection, selected classes (i.e., its constructors, instance and static methods) are wrapped at runtime in several layers, finally being of a type that the GP framework can work with. A downside to this approach is the poorer performance generally associated with reflection (some JVM optimizations cannot be applied when dynamically resolving types) which can be partly remedied using libraries like ReflectASM[4].

4 Evolutionary Art

> "Evolutionary art is a research field where methods from Evolutionary Computation are used to create works of art [...]" [dHE10, p. II].

A good overview of the field is given in [Lew08]. There have been numerous approaches at both representation and fitness function in EA, some of which will be introduced in this chapter.

Generally, EA is just another application of evolutionary computation, and it follows the evolutionary process as described in section 2.1.

EA has its origin in [Daw86], and [Sim91] was the first to focus solely on expression-based EA. After that, many artists and scientists have followed them and explored the possibilities of EA.

This section will first give an overview of the different representations used in the generation of 2D imagery (section 4.1) and then explore the different ways to determine the fitness of images (section 4.2), with special focus on automatic evaluation algorithms (section 4.3).

4.1 Representation

In the context of EA, the image is the phenotype that is evaluated. Genotype representations, on the other hand, can take any form, usually involving either a fixed size set of numbers (as in GA) or a hierarchy of expressions (as in GP). Genotypes are then transformed to phenotypes (e.g., 3D objects, pieces of music, or, as in this case, images).

[Sim91] was among the first to use an expression-based genotype for evolutionary art. He evolved trees of LISP expressions, the function including "various vector transformations, noise generators, and image processing operations, as well as standard numerical functions" [Sim91, p. 2].

[4] `https://github.com/EsotericSoftware/reflectasm`, accessed November 28, 2015.

In this way, he was able to generate complex images from relatively simple expression trees. Examples of his work can be found in [Sim91].

Through the 1990s, many adopted his approach and tried to create EA using different function sets and expression techniques.

Expression-based genotype representation is one of the most flexible representations, as it does not limit the size or structure of genotypes. Depending on the function set and parameters, however, it is slow in evolving aesthetic images, for the very same reasons.

Most expression-based systems using Sims' approach use mathematical functions and only local information to determine pixel colors, and it is often possible to recognize from the generated images which types of functions were used in their creation.

There have been other representations, including Fractals (using iterated function systems), most notably used in the Electric Sheep project[5], neural networks, images evolved by processing existing images, and lines and shapes evolved using ant or swarm algorithms [Lew08].

4.2 Fitness function

There are two basic approaches to fitness functions in EA: First, there is the interactive evaluation. It defers evaluation to a human user who can, e.g., pick preferred images or evaluate images with a numeric value. These evaluations are then used to evolve the images. This approach was used in many early implementations of EA, because it is easy to implement and the evaluation takes no computation time at all.

Then again, the evaluation takes a lot of time, because humans are slow, quickly fatigued and/or bored, so new ways to evaluate images automatically had to be found.

The limitations of interactive evaluation (not only in the field of EA) quickly brought on research in the field of automatic evaluation. One of the early approaches at automatic evaluation in EA was introduced in [BPJ94], where neural networks were trained through interactive evaluation and could later perform the evaluation without the user, even on images never encountered before. Several problems were encountered using this approach and the results were not satisfying to a degree where the neural network could be used as a replacement for the human evaluation.

It became obvious, that an automatic aesthetic measure is not a trivial problem, because it has to model a part of human behavior or preference which is not sufficiently understood yet. Two automatic image evaluation algorithms are described in section 4.3, both of which result in a slightly different "kind" of resulting image. The generated images are not quite the same, but they share some traits identifying the algorithm which helped evolve them.

In [McC05], where open problems in evolutionary music and art are discussed, one of these problems is identified to be the design of "formalized fitness functions that are capable of measuring human aesthetic properties of phenotypes. These functions must be machine

[5] `http://www.electricsheep.org/`, accessed November 28, 2015.

representable and practically computable" [McC05, p. 5]. This is exactly the problem one faces when designing and implementing automatic evaluation algorithms.

4.3 Automatic evaluation algorithms

4.3.1 Machado & Cardoso: Image complexity

In [MC98], a system to create visual art was developed, the focus being on the automatic evaluation of grayscale images based on the understanding of human aesthetic judgment at the time.

The authors identified several problems which make the evaluation of visual art challenging. Among them was the fact that visual art theory is not as advanced as in other fields of art (like music) and there is no underlying notation system for visual art.

After arriving at the conclusion "that the assessment of the visual aesthetic value of an image is directly connected to the visual image perception system" [MC98, p. 4], they go on to break down the visual value of an image into two components: Processing complexity (PC; where lower is better) and image complexity (IC; where higher is better). For both methods, image compression is used as a measure. For image complexity, JPEG image compression is used. The compressed image is compared to the original using the quotient

$$IC(I) = \frac{RMSE(I)}{Compression\_ratio(I)} \tag{1}$$

where RMSE is the root mean square error between the images, and the compression ratio is the ratio between the image sizes before and after compression. The less predictable the pixels of the image are, the greater the IC measure. Fractal image compression, which basically explores self-similarities of an image, is used to measure the processing complexity as it resembles the human image processing mechanism more closely than other compression techniques. Additionally, for PC, the human perception process (i.e., more details are observed the longer the image is being processed) is imitated by compressing and evaluating the image twice–with different compression quality settings. This is not done for IC, as JPEG compression degrades too much for high compression ratios [MC98, MC02, dHE10].

In [ESC11], different estimates for image complexity and processing complexity are used: Image complexity is defined only through the compression ratio, i.e. the predictability of the pixels, while processing complexity is expressed as the compression ratio of the genotype, so it is rather a measure of genotype complexity.

4.3.2 Global contrast factor

The global contrast actor (GCF) is a contrast measure which tries to model the human perception of contrast when looking at an image. It computes an overall contrast value by

inspecting local contrast factors at different resolutions and building a weighted average.

For this, the *perceptual luminance* L is calculated for each pixel:

$$L = 100\sqrt{\left(\frac{k}{255}\right)^{\gamma}}. \tag{2}$$

where $k \in \{0, 1, \ldots, 254, 255\}$ is a pixel value and $\gamma = 2.2$ is a gamma correction parameter.

The local contrast lc is the average difference in perceptual luminance between neighboring pixels. The average local contrast C_i is calculated for different resolutions using

$$C_i = \frac{1}{w \cdot h} \sum_{i=1}^{w \cdot h} lc_i \tag{3}$$

with w and h being the image dimensions at the current resolution. With this, the GCF can be calculated as

$$GCF = \sum_{i=1}^{N} w_i \cdot C_i \tag{4}$$

In each iteration (i.e., for N different resolutions) several pixels are combined into one superpixel, this is repeated until only a few superpixels are left. The local contrasts are then summed up using weight factors w_i as described in [MNN+05].

Images of lower contrast are considered less interesting than images with higher contrast. More details on this algorithm and its implementation can be found in [Rü13a] and [MNN+05].

5 Results

In this section, example images for different image creation and evaluation methods will be shown, to give a general overview of the results achieved in [Rü13a] without going into details.

5.1 Image creation methods

Two different image creation methods have been used in [Rü13a]. For most of the tests and experiments, arithmetic expression-based function image creation has been used. Alternatively, basic fractal expression-based image creation has been implemented. In this section, they will be compared.

On the genotype level, both methods use an expression tree. In the case of function image creation, the return value of the expression tree is a color vector, representing the color of a specific pixel, and variables (representing pixel coordinates) are incorporated into the tree which can be changed from the outside to generate different results for multiple executions of the expression tree. The fractal image creation works differently: The expression tree directly returns an image upon evaluation, encapsulating the whole fractal image creation into the primitive set. For this reason, the expression tree has to be evaluated only once.

The primitive sets used for both image creation methods contain primarily standard operations (addition, multiplication etc.), trigonometric functions (like sine, cosine and tangent), number type conversion functions, simple conditional functions and rounding functions. Additionally, the primitive set for fractal image creation contains complex functions.

Figure 1 shows example images of function and fractal image creation. Fractal image creation, with a fixed number of iterations and a fixed formula, shows less phenotype diversity, while function image creation (more examples can be found in the later section of this chapter) can compete in image complexity but is overall slower than fractal image creation.

5.2 Interactive image evolution

Figure 2 shows four example images that were generated at different stages of the evolutionary process using interactive image evaluation.

5.3 Automatic image evolution

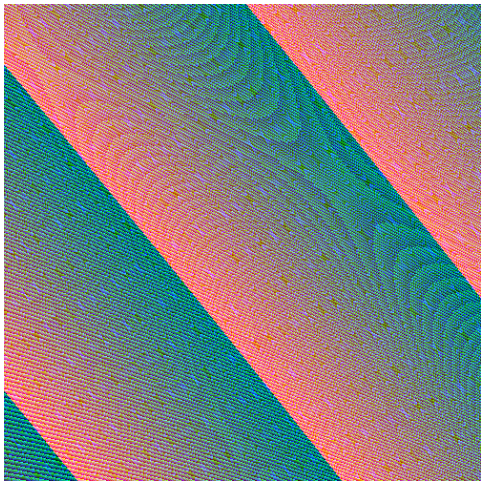
The following sections will present evolution results for different automatic evaluation algorithms.

5.3.1 Global contrast factor

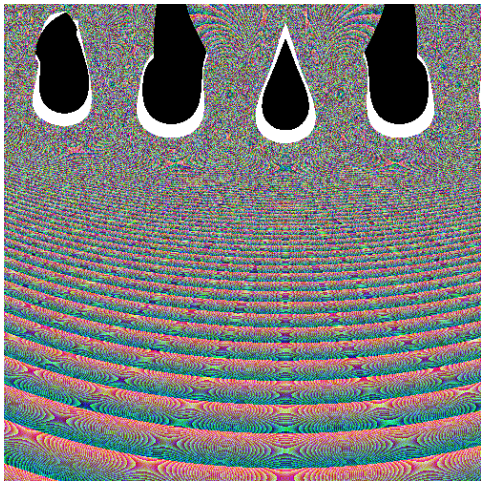
The GCF, as described in section 4.3.2, is an algorithm which evaluates the amount of detail (as perceived by a human viewer) in an image.

Figure 3 shows two example results of evolutionary runs using GCF as an evaluation measure. The results are similar in that they exhibit many sharp edges and no smooth color gradients–as expected of an algorithm that prefers local contrasts, i.e., many details.

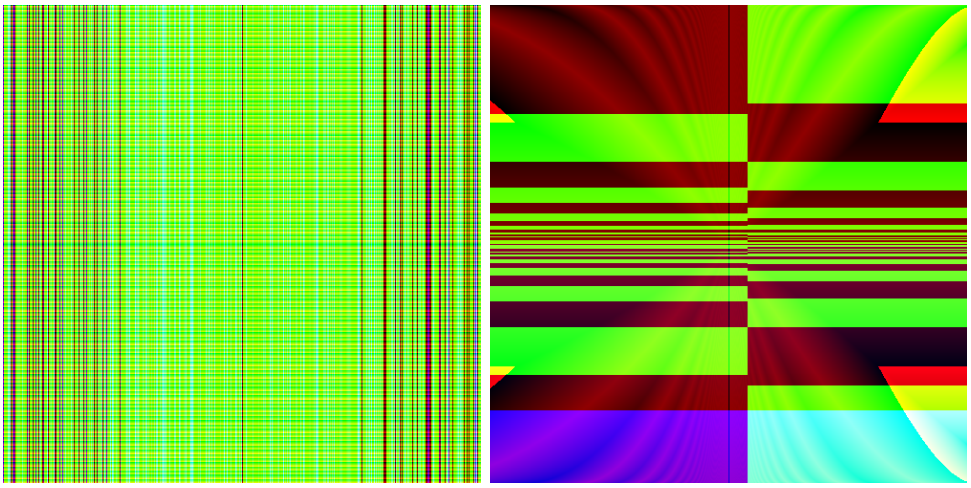
Overall, the GCF works well as a primary aesthetic measure as it favors images with many details without producing images which are as distorted or “noisy” as the image complexity measure (see section 5.3.2).



(a)

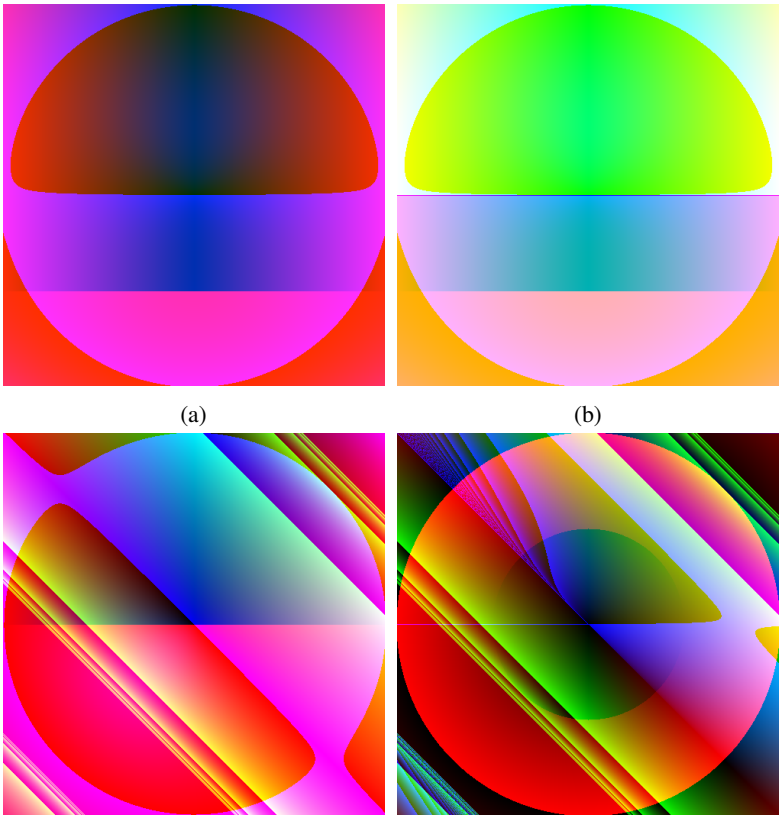


(b)



(c) (d)

Figure 1: Image examples of fractal ((a) and (b)) and function-based ((c) and (d)) image creation.



(a) (b)

(c) (d)

Figure 2: Image examples of interactive image evolution at different stages of the evolution: Initial population (a), after five generations (b), after ten generations (c) and after 15 generations (d).

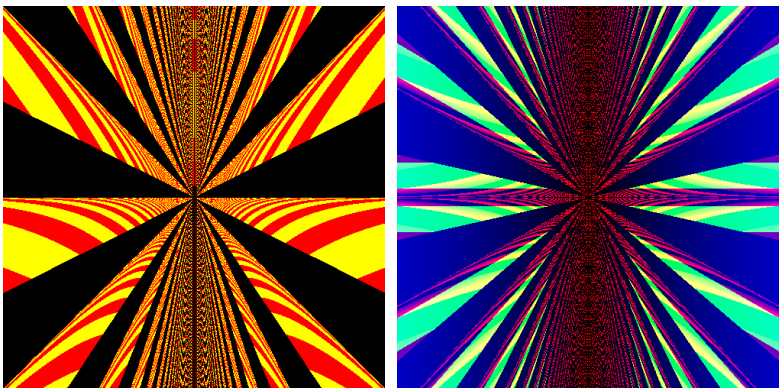


Figure 3: Example results of GCF-guided image evolutions.

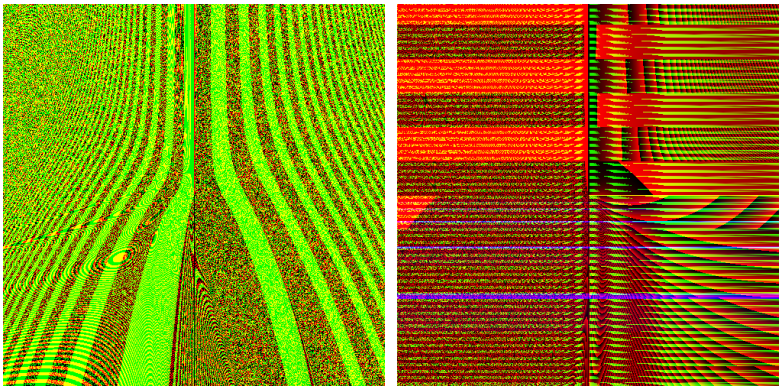


Figure 4: Example results of IC-guided image evolutions.

5.3.2 Image complexity

The image complexity measure, as described in section 4.3.1, is an algorithm which evaluates image complexity by looking at the performance of JPEG compression of an evolved image–compression size and the error in the compressed image (RMSE) are taken into account. It is part of the Machado & Cardoso aesthetic measure and will, at this point, only take the image complexity into account, ignoring the processing complexity (more details in [Rü13a]).

Figure 4 shows two example results of the IC-guided evolution. It can be seen that IC prefers images with a lot of color distortion or "noise", i.e., big (and hard to predict) changes in color between neighboring pixels, as these are hard to compress and produce images with higher complexity.

Image complexity works well as a primary evolutionary objective and produces aesthetically pleasing images, although, in later stages of the evolution, the subjective image quality degrades because of the color distortion and noise.

5.4 Multi-objective image evolution

Here, multi-objective image evolution is performed using the two aestehtic measures IC and GCF, as well as bloat control (using the genotype complexity). Figure 5 shows two examples of the multi-objective image evolution. IC is overall the stronger measure because it has greater absolute values, but the influence of the GCF evaluation can still be seen in the generated images.

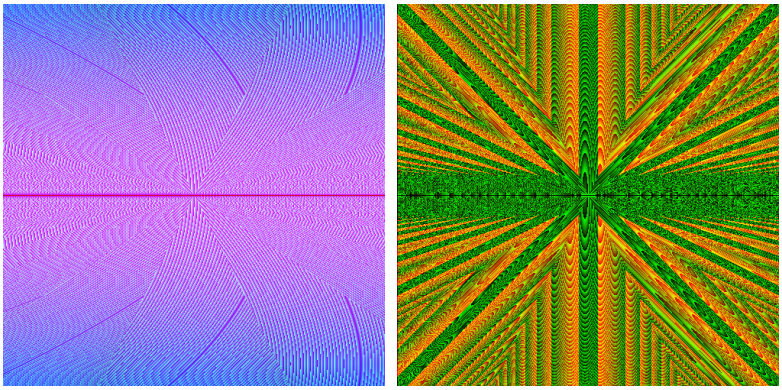


Figure 5: Example results of multi-objective image evolutions.

6 Conclusion

This paper has given an overview of [Rü13b] and [Rü13a], summarizing the most important techniques, algorithms, and results. Various tests have been performed using automatic and interactive image evolution with different combinations of evaluation algorithms using the Jpea[6] application.

The most successful automatic image evaluation algorithms introduced here were global contrast factor evaluation and image complexity evaluation. Arithmetic function image creation provides a wide variety of different types of images, even though it is limited in that it only incorporates local information when generating images.

Fractal image generation has potential to generate beautiful images, but has many drawbacks, including slow generation speed, complex coloring algorithms that need to be tuned, and a limited generalization. In short, fractal image generation has been used more successfully in GA systems than in GP systems, i.e. in scenarios where a function of fixed structure is given whose parameters are evolved. When trying to generate fractals or iterated function systems from scratch, the number of interesting solutions is lost in the vast amount of uninteresting ones.

Concluding, different existing image evaluation and creation algorithms (using different primitive sets) have been implemented and their results have been presented. It can be said that evolutionary art is still a field with much potential, especially when looking at future processor speeds and increasing number of processor cores. Automatic evaluations are useful but still do not really capture human aesthetic preferences–a fact that might not change in the near future, seeing how complex of a topic it is.

[6] `https://gitlab.com/jpea/jpea`, accessed November 28, 2015.

References

[Atm94] W. Atmar. Notes on the simulation of evolution. *Neural Networks, IEEE Transactions on*, 5(1):130–147, 1994.

[BFM97] Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz, editors. *Handbook of Evolutionary Computation*. IOP Publishing Ltd., Bristol, UK, 1st edition, 1997.

[BFM99] Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz, editors. *Basic Algorithms and Operators*. IOP Publishing Ltd., Bristol, UK, 1st edition, 1999.

[BNKF98] Wolfgang Banzhaf, Peter Nordin, Robert E. Keller, and Frank D. Francone. *Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications*. Morgan Kaufmann, San Francisco, CA, USA, January 1998.

[BPJ94] Shumeet Baluja, Dean Pomerleau, and Todd Jochem. Towards Automated Artificial Evolution for Computer-generated Images. *Connection Science*, 6(2-3):325–354, 1994.

[Bä96] Thomas Bäck. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford University Press, Oxford, UK, 1996.

[Daw86] Richard Dawkins. *The blind watchmaker: Why the evidence of evolution reveals a universe without design*. WW Norton & Company, 1986.

[dHE10] E. den Heijer and A. E. Eiben. Comparing aesthetic measures for evolutionary art. In *Proceedings of the 2010 international conference on Applications of Evolutionary Computation - Volume Part II*, EvoCOMNET'10, pages 311–320. Springer-Verlag, 2010.

[ESC11] Anikó Ekárt, Divya Sharma, and Stayko Chalakov. Modelling human preference in evolutionary art. In *Proceedings of the 2011 international conference on Applications of evolutionary computation - Volume Part II*, EvoApplications'11, pages 303–312. Springer-Verlag, 2011.

[Hol75] J Holland. Adaptation in artificial and natural systems. *Ann Arbor: The University of Michigan Press*, 1975.

[Koz92] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[Lew08] Matthew Lewis. Evolutionary Visual Art and Design. In Juan Romero and Penousal Machado, editors, *The Art of Artificial Evolution*, Natural Computing Series, pages 3–37. Springer Berlin Heidelberg, 2008.

[Luc04] Simon Lucas. Exploiting Reflection in Object Oriented Genetic Programming. In Maarten Keijzer, Una-May O'Reilly, Simon M. Lucas, Ernesto Costa, and Terence Soule, editors, *Genetic Programming 7th European Conference, EuroGP 2004, Proceedings*, volume 3003 of *LNCS*, pages 369–378, Coimbra, Portugal, April 2004. Springer-Verlag.

[MC98] Penousal Machado and Amílcar Cardoso. Computing Aesthetics. In *Advances in Artificial Intelligence*, volume 1515 of *Lecture Notes in Computer Science*, pages 219–228. Springer Berlin Heidelberg, 1998.

[MC02] Penousal Machado and Amílcar Cardoso. All the Truth About NEvAr. *Applied Intelligence*, 16(2):101–118, 2002.

[McC05] Jon McCormack. Open Problems in Evolutionary Music and Art. In Franz Rothlauf, Jürgen Branke, Stefano Cagnoni, DavidWolfe Corne, Rolf Drechsler, Yaochu Jin, Penousal Machado, Elena Marchiori, Juan Romero, GeorgeD. Smith, and Giovanni Squillero, editors, *Applications of Evolutionary Computing*, volume 3449 of *Lecture Notes in Computer Science*, pages 428–436. Springer Berlin Heidelberg, 2005.

[Mil11] Julian F. Miller, editor. *Cartesian Genetic Programming*. Natural Computing Series. Springer, 2011.

[MNN+05] Krešimir Matković, László Neumann, Attila Neumann, Thomas Psik, and Werner Purgathofer. Global contrast factor - a new approach to image contrast. In *Proceedings of the First Eurographics conference on Computational Aesthetics in Graphics, Visualization and Imaging*, volume 1 of *Computational Aesthetics'05*, pages 159–167. Eurographics Association, 2005.

[PLM08] Riccardo Poli, William B. Langdon, and Nicholas Freitag McPhee. *A field guide to genetic programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008. (With contributions by J. R. Koza).

[Rü13a] Johannes Rückert. Artificial Art: Image Generation using Evolutionary Algorithms. Bachelor's thesis, Fachhochschule Dortmund, 2013.

[Rü13b] Johannes Rückert. Reflection-basierte Genetische Programmierung am Beispiel Evolutionärer Kunst. Project thesis, Fachhochschule Dortmund, 2013.

[Rü16] Johannes Rückert. Artificial Art: Image Generation using Evolutionary Algorithms. In Britta Böckmann and Robert Preis, editors, *Schriftenreihe des Fachbereichs Informatik der Fachhochschule Dortmund, Band 2*, MV-Wissenschaft, pages 175–190. Monsenstein und Vannerdat, 2016.

[Sch75] Hans-Paul Schwefel. *Evolutionsstrategie und numerische Optimierung*. PhD thesis, Technische Universität Berlin, 1975.

[Sch77] Hans-Paul Schwefel. *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie – Mit einer vergleichenden Einführung in die Hill-Climbing- und Zufallsstrategie*. Birkhäuser, Basel, 1977.

[Sch81] Hans-Paul Schwefel. *Numerical Optimization of Computer Models*. John Wiley & Sons, Inc., New York, NY, USA, 1981.

[Sim91] Karl Sims. Artificial evolution for computer graphics. *SIGGRAPH Comput. Graph.*, 25(4):319–328, July 1991.

[WM97] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *Trans. Evol. Comp*, 1(1):67–82, April 1997.