Concurrent Computational Environments for Generative, Abstract Geometric Design

Prof. Nikitas M. Sgouros
Department of Digital Systems, University of Piraeus, Piraeus, Greece
www.mywebsite.com
e-mail: sgouros@unipi.gr

Abstract

Abstract geometric forms offer infinite design opportunities as they are not subject to the constraints of their representational equivalents. We explore the creation of abstract designs based on abstract art methods that generate and evolve rich sets of concurrent relations on the basis of form, colour, movement and rhythm and apply them onto various interacting shape configurations. Based on the need for concurrency and interaction on a massive scale we argue for the creation of a new generation of concurrent computational environments that can support such artful activities. We describe such an implementation based on Elixir a functional and concurrent programming language. Furthermore, we describe and give examples of our methodologies in creating abstract designs based on our implementation.

1. Introduction

Abstract geometric forms refer to visual forms that do not represent real-world objects. The lack of correspondence between such forms and real-world equivalents can create infinite design opportunities as the analogs of real-world constraints can be ignored. However, the design process in this case becomes more complex due to the sparsity of tools and methods that can aid the designer. Since there are no real-life analogs to guide one's creative efforts, the design of abstract forms, as realized in various approaches in abstract art, explore the generation and evolution of rich sets of concurrent relations on the basis of form, colour, movement and rhythm and apply them onto various shape configurations. Such visual experiments seek to provide aesthetically interesting and stimulating experiences to the observer. In the case of digital media, the creation and evolution of such rich sets of relations both within and across different shape configurations in real-time is computationally expensive due to their number and the large sets of individual shapes and/or points that each one of them involves. While existing visual computing architectures (e.g. GPUs) support parallel processing of various image elements in isolation, they are not very efficient in implementing parallel interactions between these elements. Therefore, if we seek to support abstract

art methodologies in design, we need to construct novel computational environments able to support concurrent processing between various visual elements within and across all image design levels; configuration, shape or point.

In this paper we describe a series of visual experiments on the creation and evolution of composite, two-dimensional abstract forms based on our digital interpretation of abstract art methodologies. We implement each form as a particular, animated configuration of various parametric geometric shapes. In computational terms we decompose each configuration into a set of concurrent processes corresponding to its points that can exchange messages with each other. In order to achieve an optimal level of concurrency we have implemented our experimental design system in Elixir [7], a functional and concurrent programming language.

2. Generating Configurations of Abstract Geometric Forms

The design of abstract forms has to take into account the specific ways by which abstract content is perceived by the observer. More specifically, our brain has evolved to act as a pattern recognizer and our visual system has been trained to recognize patterns that correspond to objects occurring in nature. Representational forms exploit these capabilities by employing well-trained, bottom-up visual processes, such as line, corner or contour detection to recognize objects in the depicted content and therefore extract meaning from an image [4]. This is not the case with abstract forms which do not correspond to natural objects and therefore cannot exploit bottom-up processes in perceiving the image. Instead perception of abstract content relies on top-down processing that associates visual input with our imagination, experience and/or emotions [1]. In this sense, the perception of abstract visual forms has significant similarities with music perception something that has not escaped the attention of important abstract artists such as Kandinsky [2] or Mondrian [3]. Freeing the image from form and colour constraints characterizing natural objects and from viewing conventions such as perspective invites the observer to actively construct associations that link visual input with his or her own experience during the interpretation of abstract content.

2.1 Seeding

Our approach to abstract design relies on the continuous generation of shape configurations that realize symmetrical and rhythmic associations. We refer to the process of creating a new shape configuration as *seeding* because it generates an initial composition of shapes that will be modified during: (i) the execution of all the animations associated with the shapes involved in the seed, (ii) its interaction with the rest of the configurations that are currently active.

Each seed is structured as a sequence of levels, where each level consists of a set of shapes belonging to the same geometric family with randomly varying parameters of construction. Currently, possible families of shapes at each level include line segments, ellipses, Lissajous curves, epi- and hypo- trochoids [8]. For example, we can specify a configuration as a composition of three levels where the first level L1 consists of a single ellipse, the second level L2 consists of line segments with random lengths and

directions, while the third level L3 consists of epitrochoid curves with randomly chosen parameters of construction. Each shape at the previous level provides the points of reference for placing the shapes at the next level. Reference points are computed at a regular spacing across the circumference of the current shape. The spacing can be specified at compile time or it can be determined by the system or the user at run-time for each level. Continuing our example configuration, we can specify that each line at L2 will provide the point of reference for the generation of an epitrochoid curve every 20 points along its length. Therefore if a specific line has a length of 100 points it will generate five epitrochoid curves along its length. Figure 1 depicts two example configurations each one consisting of two levels. Figure 1 (left) shows an elliptical segment at level 1 that hosts four epitrochoid curves at regular distances at level 2 creating an ornamental pattern. Figure 1 (right) shows a vertical line at level 1 that forms the locus for a set of equidistant horizontal lines at level 2 along the vertical line. The goal here is to create rhythmic associations between the horizontal lines.

All positions in a configuration are carried out in a 3D logical coordinate system common to all of them. Viewing employs a screen coordinate system implemented as an integer grid of pixels. Every position in the logical coordinate system is mapped to the nearest pixel in screen coordinates. A variety of mapping techniques can be used (e.g., perspective, orthographic projection) and they can be specified at a configuration, shape or point level. For example, we can specify that all shapes at level L1 of a configuration will be orthographically projected while those at level L2 will be mapped with perspective.

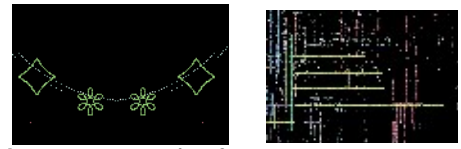


Figure 1: *Example of 2-level shape configurations.*

Shapes belonging to the same level initially share the same colour. Furthermore, shapes belonging to the same level share the same animation pattern. Such an animation pattern can be defined: (i) relative to the reference point of each shape in the level (e.g. rotate around the current position of your reference point), (ii) using absolute positioning in the screen (e.g. move towards the screen center).

In the first case, if the reference point moves because the shape it belongs to is animated as well, then its associated shape will follow it before executing its own animation step. For example, all L2 shapes at Figure 1 can rotate with the same angular speed around their reference point or they can translate in parallel directions. Furthermore, if these animations have been specified relative to the animation patterns of L1 and L1 shapes move towards the center of the screen, then all L2 shapes will first follow their reference points in their translation towards the center and then execute their own animation step.

Such an organization creates a wealth of relations in a seed. The observer is able to perceive its level structure based on shape similarity, colour sharing and the similar animation patterns executed among all the members of a level. Furthermore, the ability to establish relative animation patterns between levels allows each configuration to move

and therefore to be perceived as a whole. Rhythmic relationships are established through the regularity of the placement of each shape based on its reference point. An important source of visual complexity comes from the novelty generated when shapes combine in the configuration especially when their reference points are close to one another. For example Figure 2 (left) depicts how three Lissajous curves drawn on the periphery of a circle combine to create a novel shape configuration that inherits some of the harmonic motion properties encoded in the Lissajous curves with the symmetry of a circle. Figure 2 (center) depicts a 2-level configuration of epitrochoids and hypotrochoids, while Figure 2 (right) depicts a 3-level configuration of an elliptical segment (L1), a set of lines (L2) and a set of epitrochoids (L3).

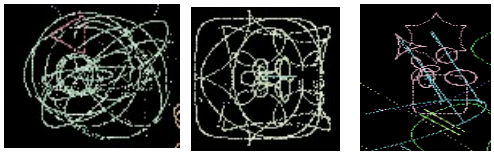


Figure 2: *Visual examples of various shape combinations in configurations.*

2.2 Configuration Dynamics

We view a configuration as a continuously evolving form based on: (i) *internal dynamics* such as the animations executed by the population at each level or the extinction of parts of the configuration, (ii) *local interactions* such as the resolution of conflicts related to the simultaneous mapping of points belonging to different configurations at the same pixel on the screen, (iii) *global constraints* imposed by the system on all configurations such as the maximum number of points that can be mapped onto a pixel or the maximum lifetime of a configuration.

In terms of internal dynamics, animations affect either the colour or the position of each point in a configuration and, as noted in the previous section, they can be either absolute or relative to their association points. In addition, for each point belonging to a configuration there is a probability of extinction that is common for all points belonging to the same level. This is computed every time the point is mapped onto a pixel. If extinction happens the process associated with the point is terminated and the specific point will never be drawn again. For example, we can specify that all points at level L1 in a configuration may have an extinction probability of 20%. This means that whenever a point in L1 is redrawn there is a 20% probability that this will not happen and the specific point will be erased from the configuration.

With regards to local interactions, the system operates under a set of rules governing the interaction of pairs of configuration points that map onto the same pixel. These rules specify a hierarchy with which such conflicts are resolved based on dominance relations between configurations. In our current visual experiments, each configuration has a dominance level associated with it. In case of conflicts, its points will be drawn over points coming from configurations having a lower dominance level. In this case, points with lower dominance levels become extinct as well. Furthermore, both points in a pair become extinct if they conflict and their dominance levels are equal. The designer can specify alternative interaction rules, for example, mixing the colours of points with the same dominance levels instead of erasing

them. We have decided on the particular set of rules in order to generate novel visual forms in cases where configurations come into contact or have occlusion relations.

Furthermore, local interactions are used to create symmetrical relations in an image. In this case, each point in a configuration creates a set of new points containing copies of itself at positions implementing various translational, rotational or point symmetries. These points are then projected onto their nearest pixels generating a wealth of symmetrical relations.

Finally, global constraints can impose limits either at a pixel or at a configuration level. In the first case, these may involve an upper limit on the possible number of points that can be projected onto any given pixel. Whenever this limit is exceeded all points beyond this limit are culled based on the time of their first projection on this pixel, with older ones culled first. Temporal culling of points causes consistent shape distortions that combined with symmetry can lead to interesting visual effects as Figure 3 indicates. Furthermore, global constraints may be used to specify the background colour of pixels in which no points are projected at some point in time.

2.3 Image Generation

Image generation is essentially an exploration process that proceeds with the continuous generation of new seed structures that are created either automatically by the system or manually by the user at chosen positions. Every new seed interacts with whatever is already projected on the same pixels on the screen creating novel visual combinations. Figure 4 (up) provides an example of the evolution of an image built with 2-level configurations, one of which is depicted in Figure 2 (center), while Figure 4 (down) provides the evolution of 2-level configurations using animated seeds similar to the ones in Figure 1(right).

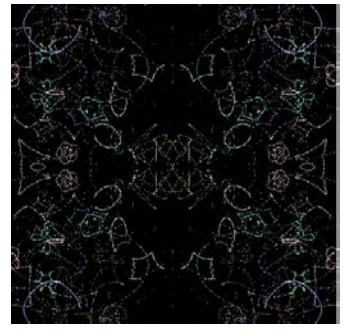


Figure 3: *Temporal point culling creates consistent gaps in closed shapes.*

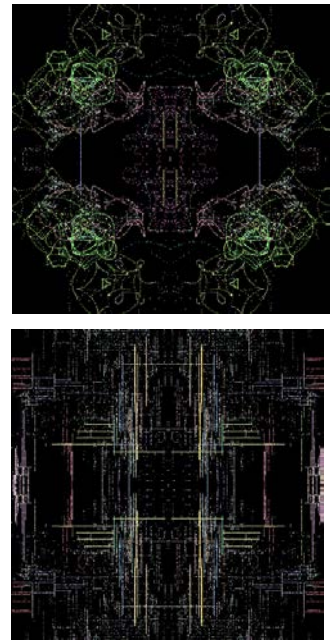


Figure 4: *Evolution of 2-level configurations.*

Furthermore, Figure 5 provides two examples of the evolution of 3- and 4-level configurations. In both figures 4 and 5 all configurations generate a pmm

symmetry cell [5].

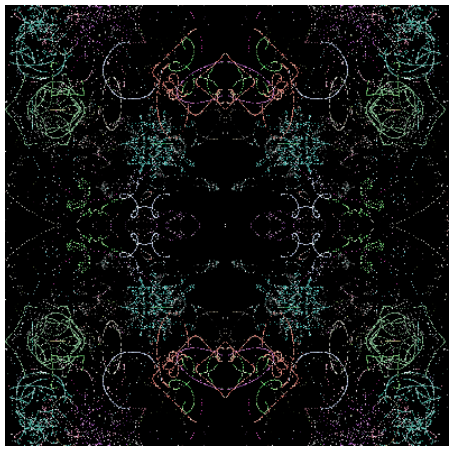
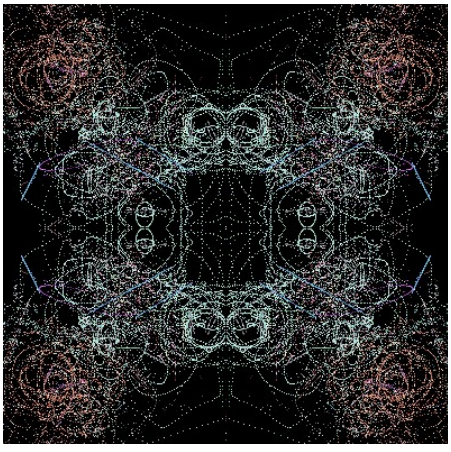


Figure 5: Evolution of 3-level (up) and 4-level (down) configurations.

3. Computational Infrastructure

3.1 Design Philosophy

The goal of generating and evolving large numbers of spatiotemporal associations concurrently between different locations in an image necessitates the development of a computational framework that can support the



complexity generated in this type of visual abstraction. Furthermore, the need for each visual element to both remain a part of a larger configuration while acting autonomously and interacting with the rest of the content, requires each element to have a conception of state and ways of manipulating it along with the ability to communicate with its environment. For example, evolving an image such as the one in Figure 5 (down) requires approximately 200000 concurrent processing elements able to exchange messages with each other while at the same time each one managing its own state. Thus, the computational framework upon which such a system can be built must support massive, real-time concurrency and interaction while at the same time remaining scalable (i.e., able to deal with content of variable complexity) and reliable (i.e., either avoid or recover gracefully from execution errors). In more technical terms, this means that we have to move towards systems that can exploit concurrency at a CPU level based on the exploitation of multiple CPU cores rather than at a GPU level where interaction between processing elements is more cumbersome.

One way of fulfilling these requirements is to structure all visual computations on a set of computationally simple, concurrent processes. Simplicity allows these processes to execute fast and reliably, while consuming minimal amounts of resources. To this end, we have opted to implement our visual elements both at a configuration and at a pixel level as concurrent stateful processes that share no memory and communicate via asynchronous messages. This design avoids synchronization pitfalls (e.g.. deadlocks) and eliminates significantly complex

management of shared resources (e.g. via locks, mutexes, or semaphores). As a result, operation and interaction between concurrent elements is much simpler to develop and understand, and execution problems of any sort remain isolated to the affected processes and do not spread to the rest of the system, thus providing a high degree of reliability.

In terms of implementation we found that all these requirements are satisfied with Elixir [6], a concurrent, functional language built on top of Erlang. Erlang is another programming language with a runtime system that has built-in support for concurrency and distribution [7]. The basic concurrency primitive in both languages is called an Erlang process (which is different from operating system processes or threads). Common Erlang/Elixir systems can run millions of such processes. Not all of them may run on a simple computer with the ability to run them in different machines in a cluster. The Erlang virtual machine, referred to as BEAM, uses its own schedulers. These distribute the execution of processes over the available CPU cores, thus parallelizing execution as much as possible and optimizing the use of CPU resources such as various memory caches. BEAM is pre-emptive. It gives a small execution window to each process and then pauses it and runs another process. Because the execution window is small, a single long-running process can't block the rest of the system. The programmer can select whether to synchronize a set of concurrent tasks by waiting for the execution of all of them before proceeding further or execute them asynchronously. The second option exploits more fully the concurrency in the system and this is the option we have chosen to follow in our experiments.

3.2 Processing Elements

Our computational model supports the various decompositions taking place during image generation and results in spawning two kinds of concurrent processes, *cpoints* and *pixels*. Both types of processes are uniquely addressable through their registration in a global register of all processes running on our system. This makes message exchange between these processes very efficient.

Cpoints are concurrent computational processes corresponding to the points of a configuration. We implement each cpoint as a separate stateful process registered with a unique id. Each cpoint is uniquely addressable based on the following scheme: (i) We label each configuration with a unique id. (ii) when it is decomposed into shape levels, each shape in a level is also labeled with a unique id generated based on the id of its configuration, its current level and the order of appearance on this level. (iii) Each shape is finally decomposed into an ordered set of points we refer to as *cpoints*. This set is ordered based on the sequence through which each point is generated in the shape. For example, we model a line segment AB containing n points as an ordered sequence of points where point A has order 1 because it is the first point in the sequence and B order n since it is the last. The order of every other point is between 1 and n and increasing as it approaches B. For example, the label for the cpoint modelling point 100 in line segment AB belonging to shape #2 at level L2 of configuration #1 is labelled as 1_L2_2_100.

As a process each cpoint is responsible for determining its current position and colour. The position of each cpoint is

expressed in a global logical 3D coordinate system common to all configurations. Every cpoint contains a sequence of animation functions that are executed in order whenever it is drawn to determine its next position and colour. If the specific point is animated relative to the reference point of its shape then it inherits all the animation functions from that reference point and the sequence of inherited functions is executed before the animation functions of the current cpoint. This allows our configurations to execute hierarchical transformations similar to the ones applied in articulated mechanisms (e.g. animation of a human skeleton). Furthermore, these animations can be extended to colour as well. For example we can specify to colour the current cpoint with a random variation of the complementary colour of its reference point.

Visually, the preemptive nature of the scheduler used for executing cpoint processes along with the limited number of CPU cores at our disposal in a specific system can create interesting indeterminacy in the resulting image. For example when a shape is animated not all of its points may move synchronously. In implementation terms, we can synchronize all the cpoint processes that correspond to the points of an animated shape and wait for all of them to execute their animation step before generating the next image frame and this can be done for all configurations. Therefore we can generate `correct' animations although waiting on the completion of a large number of concurrent processes will incur a time penalty during frame generation. Alternatively, we can generate a new frame after we have issued asynchronous drawing messages to all our cpoints. This means that in the resulting frame not all cpoints may have moved to their new positions. Figure 6 provides an example of such a situation during animation of the pattern of Figure 1 (left), where the points in the elliptical segment do not move in lockstep during their translation downwards. Because the scheduler assigns a small execution window to all processes, animation will not block and in a very short time all cpoints will have moved to their new positions. Therefore, all temporary shape distortions will not be significant. In the meantime we may get interesting and unexpected visual effects in our animations something that is quite welcome in our design approach.

Figure 6: *Asynchronous execution of animation steps.*

Furthermore, each cpoint contains its projection function that computes the pixel position on which it will be mapped on the screen, along with its extinction probability. Finally each cpoint sends and receives messages through its own messaging queue. As Elixir allows these messages to contain a rich set of data (e.g. lists, map structures, strings, functions) the designer has significant degrees of freedom in implementing all sorts of communication schemes between cpoints.

If a cpoint corresponds to a reference point in a configuration, it spawns a new generation of cpoints that correspond to the points of the shape at the next level. Therefore a cpoint can carry the seeds for other forms in the form of generative functions that can be triggered either at specific time points or as a result of its interaction with the environment. Finally,

each cpoint decides whether it is dead or alive based on its probability of extinction and this computation takes place whenever it is notified to draw itself by the pixel it maps onto.

We model the screen as an integer grid of pixels. We implement each pixel as a separate stateful process with a fixed position and a current colour. During its lifetime each cpoint continuously computes its current position and maps it onto the nearest pixel. Therefore, every pixel becomes a locus for cpoints of different configurations that project onto it. Each pixel manages its list of cpoints by: (i) computing dominance interactions between cpoints, (ii) computing its colour based on a linear blending of the colours of all alive cpoints in its list, (iii) notifying all its cpoints to execute their animation steps by sending to them appropriate `draw' messages, (iv) erasing cpoints from its list that have become extinct, (v) sending its colour information to other pixels in order to create various symmetries in the image.

As a result, each pixel process acts as a coordination hub for the execution of all cpoint processes in its list. In particular, whenever a pixel receives a message to draw itself it executes all of the above functions by communicating with its cpoints using asynchronous messages. In order to expedite this process at the start of its drawing procedure each pixel asks every one of its cpoints whether it is alive or dead. Dead cpoints are removed from its list. It then computes all dominance interactions between its remaining cpoints so that the list of its available cpoints can be pruned even more before the remaining operations can be performed on them. These steps reduce both the overall number of processes active and the overall number of messages exchanged between processes so that computational load can be minimized.

The overall drawing algorithm runs in a continuous loop during which asynchronous `draw' messages are sent to all the pixel processes. These causes the pixel processes to send asynchronous `draw' messages to all of their cpoints. Furthermore, at each step the system may create new configurations thus providing continuous visual stimulation to the user. Currently, these new configurations are created by the system at random points on the screen or by the user by clicking at a desired point. We enable user interaction with our system using the Phoenix web framework [9] that employs the Erlang virtual machine and Elixir to handle multiple user connections.

4. Discussion & Related Work

Visual abstract art works and methodologies have created significant interest in relation to their interaction with the viewer. It has been argued [1, 4] that by recruiting simple geometric forms that have a meaning of their own, abstract art is able to create associations to the user that activate our emotions, imagination and creativity. Our research seeks to explore form generation and interaction inspired by abstract art methodologies by building design environments that exploit large-scale concurrency and interaction among computational process. Therefore, this research seeks to extend artistic development systems such as Processing [10] or p5.js [11] that are not specialized in supporting massive concurrency and interaction in multicore CPU environments.

In terms of content we are primarily interested in the visual possibilities created from the interactions between moving, abstract geometric configurations and in the ways with which such interactions can be combined with symmetry. As a composition method, we believe that symmetry can play a major role in the design of abstract forms because it: (i) multiplies visual relations among preexisting forms and thus can enrich the aesthetic attractiveness and the `meaning' attributed to the resulting artifact, (ii) is easily perceived [12] and satisfies the need of our vision system to detect regularities in its input.

In this respect our approach differs from artificial life or generative art approaches that evolve forms based on a set of explicit rules and/or several fitness criteria [13]. In our approach, movement is the main method through which forms change since this is how all interactions take place. Therefore, our approach relates more strongly with abstract art methodologies such as action painting [14] which emphasize action over form in the creation of an image.

5. Conclusions & Future Work

We have described our experiments on the creation and evolution of composite, two-dimensional abstract forms. Our future work will focus on further developing our web interfaces in order to allow our computational environments to be broadly accessible to interested users. Furthermore, we seek to explore the use of our abstract iconography in AR environments and/or games.

6. Acknowledgements

This work has been partly supported by the University of Piraeus Research Center.

7. References

[1] E. Kandel, Reductionism in Art and Brain Science, Columbia University Press, 2016.
[2] W. Kandinsky, Point and Line to Plane, Dover, 1926.
[3] E. Tosaki, Mondrian's Philosophy of Visual Rhythm - Phenomenology, Wittgenstein, and Eastern thought, Springer, 2017.
[4] S. Zeki, Inner Vision: An Exploration of Art and the Brain, Oxford University Press, 2000.
[5] D. Schattschneider, The plane symmetry groups: Their recognition and notation, The American Mathematical Monthly, 85 (6): 439–450, 1978.
[6] https://elixir-lang.org/
[7] https://www.erlang.org/
[8] J. D. Lawrence, A Catalog of Special Plane Curves, Dover, 1972.
[9] https://www.phoenixframework.org/
[10] https://processing.org/
[11] https://p5js.org/
[12] J. Wagemans, Detection of Visual Symmetries, Spatial Vision, Vol. 9, No. 1, pp. 9-32, 1995
[13] A. Dorin, J. McCabe, J. McCormack, G. Monro, A. M. Whitelaw, A Framework for Understanding Generative Art, Digital Creativity Vol. 23, Nos. 3–4, pp.239-259, Taylor & Francis, 2012
[14] Rosenberg, H., The American Action Painters, Art News, 1, 1952