INTERNATIONAL CONFERENCE ON LIVE INTERFACES

evolver: an audiovisual live coding performance

Alo Allik

Queen Mary University of London
a.allik@qmul.ac.uk

Abstract. This improvised audiovisual performance explores enhanced live coding as a strategy for real time multimodal synthesis and composition. The real time decision-making process of the programmer-performer is informed and aided by interactive machine learning, artificial intelligence and automated agent algorithms. These algorithms are embedded in a network-based distributed software architecture of a multimedia performance system called *evolver* which is comprised of computer graphics, sound synthesis and algorithmic composition clients. The system facilitates human-computer interaction through live coding during performances to create extemporized immersive multimodal experiences for audiences. The multimodal content during performances is created with reactive artificial life algorithms, evolutionary sound synthesis, machine listening and music analysis. Autonomous agent systems, audio feature extraction and linked semantic data formats help the performer cope with the complexities of multimedia performance environments.

Keywords: Live coding, audiovisual performance, computer music, computer graphics, evolutionary algorithms, Semantic Web

Introduction

We live in a cultural environment in which computer based musical performances enhanced by multimedia have become ubiquitous. Particularly the use of laptops as instruments is a thriving practice in many genres and subcultures. The opportunity to command the most intricate level of control on the smallest of time scales in music composition and computer graphics introduces a number of complexities and dilemmas for the performer working with algorithms. Writing computer code to create audiovisuals offers abundant opportunities for discovering new ways of expression in live performance while simultaneously introducing challenges and presenting the user with difficult choices. There are a host of computational strategies that can be employed in live situations to assist the performer, including artificially intelligent performance agents who operate according to predefined algorithmic rules. This performance uses a software system for real time audiovisual improvisation and composition in which live coding as a computational strategy for audiovisual laptop performances is explored. The features of the framework intend to enhance the live coding practice by analysis of traditional music, evolutionary computing, reactive computer graphics and linked data technologies.

System architecture

The basic architecture of the *evolver* environment consists of

- algorithmic composition libraries developed in the SuperCollider programming environment
- a precompiled graphics application developed in C++ using the Cinder library
- a CouchDB JSON-LD database
- a Semantic Web ontology that describes the synthesis environment

The name of the performance environment is derived from a method of evolutionary sound synthesis which provides the majority of sonic material. The evolutionary algorithm enables evolving large populations of complex synthesis graphs, either in real time or for later reuse. The structure of the evolutionary synthesis process is described in a light-

weight OWL ontology, while the graphs are stored in a CouchDB database[1] and linked to the ontology using JSON-LD[2], a semantic extension of the standard JSON format. The computer graphics component implements a 3-dimensional world of cellular automata that operates in parallel as a self-organising map of audio analysis vectors in Cinder[3], an open source OpenGL library in C++.

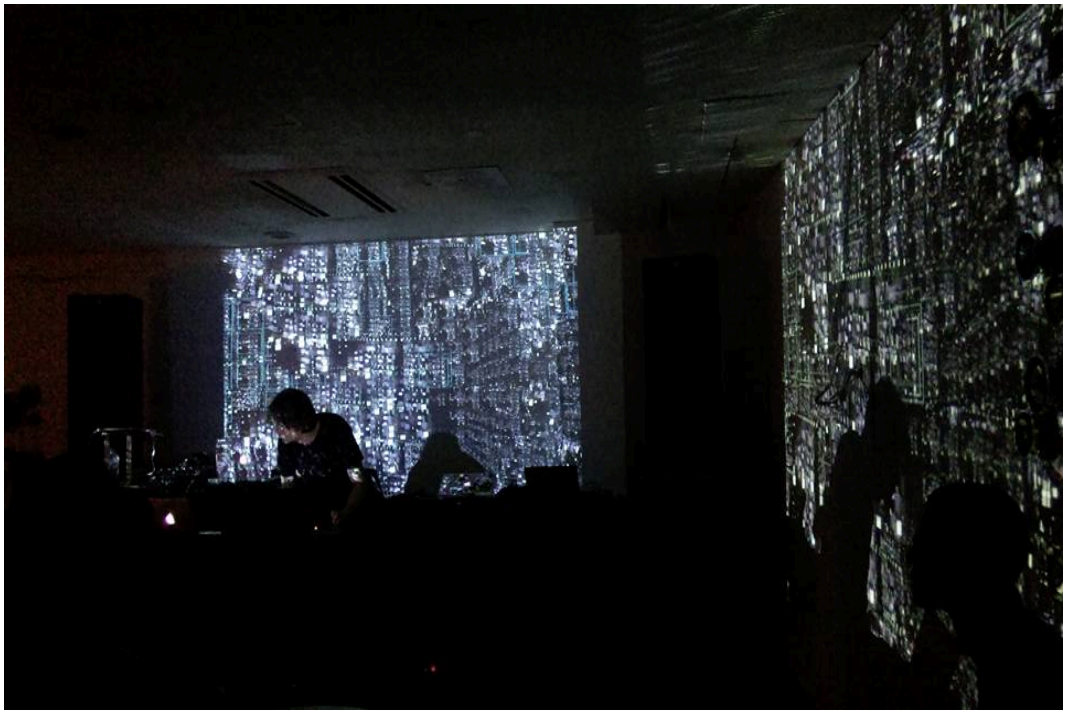

Figure 1. Performance with the evolver system

Gene Expression Synthesis

Gene expression synthesis (GES) (Allik 2014) is a way to evolve sound synthesizers in computer code. These synthesizers are computer programs that produce sound when executed. Gene expression synthesis uses the methods of gene expression programming proposed by Candida Ferreira (Ferreira 2001). In gene expression programming the popuation of candidate solutions are encoded as linear strings and then decoded into tree structures representing computer programs. Each candidate solution defined as a chromosome consists of codons - elemental units of GES - that can be either functions or terminals. The translation from genotype to phenotype follows a simple, breadth-first recursive principle: as the codons of a gene are traversed, for each function encountered, the algorithm reserves a number of following unreserved codons as arguments to that function regardless whether they are functions or terminals. Once decoded as an executable program, each candidate solution in a population is subjected to a fitness evaluation that determines how well it performs at solving a target problem.

The computer programs evolved with GES are sound synthesis graphs in this case implemented in the SuperCollider programming environment[4]. Each solution generates a SuperCollider SynthDef object. The functions in a GES chromosome are Unit Generators, sound generating functions that serve as basic building blocks of synthesis graphs. Fitness is primarily evaluated in terms of a distance metric of features from target audio, while secondary methods are

[1] http://couchdb.apache.org
[2] http://json-ld.org
[3] https://libcinder.org
[4] http://supercollider.github.io

used to ensure the structural and functional integrity of each synthesis graph as well as balancing factors between resource efficiency and graph complexity. The main fitness function measures the distance of audio feature vectors between the candidate synthesizer and a target sound, which is selected by the user depending on the context of the experiment. Resource efficiency measure has been implemented in order to imitate the condition of limited resources of natural selection, so each candidate solution is assigned a CPU usage value measured during the execution of the synthesizer. To counteract the tendency towards simplicity, a conflicting fitness pressure is introduced to encourage structural complexity in the form of rewarding greater nesting depth. This way, the complexity can be maintained in populations, while still encouraging resource usage effectiveness. Once each individual has been assigned a fitness value, the population is subjected to various standard genetic operators, including **replication** (weighted random selection of individuals for next generation according to fitness), **mutation** (a random change of a single codon value based on a user defined parameter), **transposition** (copying of codon sequences to different locations on a chromosomes), and **recombination** (exchange of genetic material between chromosomes).

Structured data storage

Due to large volumes of data potentially involving thousands of candidate solutions deemed suitable for performances, GES synthesizers with their associated metadata are stored in a CouchDB database as JSON data structures. This includes all the data necessary to reconstruct each synthesizer for use during a performance or as sources for subsequent evolutionary synthesis experiments. The *evolver* system integrates a customised SuperCollider live coding environment with a lightweight OWL ontology that enables representation of GES synthesizers on the Semantic Web. The synthesizer data that is communicated between CouchDB and SuperCollider is expressed in terms of the GES ontology using JSON-LD (Lanthaler and Gütl 2012). Listing 1 shows a fragment of a GES synthesizer data structure in JSON format as stored in CouchDB.

```
{
   "@context": { "ges": "http://geen.tehis.net/ontology/" },
   "_id": "1a75f4f87bdcac24b6ba5fc25c003ab2",
   "@type": "ges:Synth",
   "ges:environment": { "@type": "ges:Environment", "ges:headsize": 24, "ges:numgenes": 1, "ges:linker": {
      "@type": "ges:Function", "ges:name": "*", "ges:class": "AbstractFunction"
      }
   },
   "ges:defname": "gep_gen000_061_141212_225728",
   "ges:genome": [ "LFPar", "LFGauss", "SinOsc", "v", "PMOsc", "e", "j", "a", "c", "a", "a", "f", "g", "b"],
   "ges:features": {
      "ges:centroid": {
         "ges:mean": 526.07188020057,
         "ges:std_dev": 161.03829149232
      },
      "ges:flatness": {
         "ges:mean": 0.032055785092016,
         "ges:std_dev": 0.017184103858522
      }
   }
}
```

Listing 1.Fragment of a GES synthesizer JSON structure

The *evolver* environment

The gene expression synthesis algorithm can be used for isolated experiments to generate populations of synthesizers for reuse during performances. However, the textitevolver environment enables the performer to evolve and play synthesizers live on stage. The semantically enriched live coding environment aids the performer in the decision making process when selecting synthesizers from the database or evolving them in real time. The synthesizers are classified according to audio feature vectors and different maps are created to visualize the distribution of feature vectors from different perspectives. For example, a plot of spectral flatness - how noise- like a signal is - against spectral centroid -

how ”bright” the sound is - gives an idea about the characteristic of each synthesizer. This can be further aided by classifying synthesizers with a self-organised 2-dimensional map of MFCC vectors. The performer can make informed selections of synthesizers either for use in the real time live coding composition process or as source material for real time evolution by going through the iterations of the GES algorithm.

One of the performance strategies involves selecting 2 chromosomes from the database to serve as source material for on-stage experiments. These 2 chromosomes are then subjected to genetic operators, including recombination, which involves creating 2 new individuals by exchanging genetic material between the initial 2 chromosomes. The population can be subsequently grown by doubling the number of individuals each generation, evaluating the fitness of each, classifying them using the features and the existing data and selecting which ones to use in the performance based on this information. The evolved synthesizers can be used in a number of different ways as compositional elements. The target sound towards which the algorithm is converging is selected according to context. For example, percussive sounds are more suitable if the GES synthesizers are used in a dance music context to fill rhythm patterns, whereas continuous drone-like sounds are more suitable for building ambient soundscapes.

Additional Information

There are a number of existing examples of the system in use in various performance contexts:

- an experiment with evolving synthesisers in real time: https://soundcloud.com/tehis/evolver00
- a live algorave performance: https://soundcloud.com/user-665948413/osaka-live
- an example of evolver in audiovisual context: https://vimeo.com/75750212

References

Allik, A. 2014. “Gene Expression Synthesis.” In *Proceedings of the ICMC/SMC, Athens, 14-19 September*.

Ferreira, C. 2001. “Gene Expression Programming: A New Adaptive Algorithm for Solving Problems.” *Complex Systems* 13 (2): 87–129.

Lanthaler, M., and C. Gütl. 2012. “On Using JSON-LD to Create Evolvable RESTful Services.” In *Proceedings of the 3rd International Workshop on RESTful Design at WWW2012*.