Extending Context Free to Teach Interactive Evolutionary Design Systems

No Author Given

Sydney University, Sydney NSW 2006, Australia,
[rob|kaz]@arch.usyd.edu.au,
http://web.arch.usyd.edu.au/~rob

Abstract. This document presents a novel approach to teaching generative design systems by extending a design grammar to support parametric and evolutionary design. We present some of the problems that design students have learning about generative systems, describe our solution to providing students with a progressive learning experience from design grammars, through parametric design, to evolutionary design. We conclude with a discussion of the benefits of our approach and some directions for future developments.

1 Introduction

An understanding the principles of *generative design systems* is an important component of any modern education in computer-aided design. The application of generative design technologies to solve complex real-world design problems has come to public attention with the development of the "bird's nest" stadium for the Beijing Olympics. Generative design systems are also increasingly finding uses in the "mass customisation" of goods, where they can be used to produce custom designs for anything from clothing to jewellery to housing. Technologies used to create generative design systems include a wide range of computational processes inspired by nature including cellular automata, flocking systems, and evolutionary systems.

1.1 Teaching Generative Design Systems

As part if the Bachelor in Design Computing at the University of Sydney, we have been teaching generative design systems for more than five years, to both undergraduate and graduate students. The curriculum for the teaching programme introduces of a number of important concepts including; Expert Systems, Design Grammars, Parametric Design, and Evolutionary Design.

In the past we have taught these different concepts using different software packages but we found these teaching tools to be problematic. For example, a earlier classes we introduced students to evolutionary design by using them to solve a series of layout and space efficiency problems. The genetic algorithm in these problems could be modified to some extent (population sizes, mutation/crossover

rates, selection algorithms, etc.) but the problem space was always fixed. The barrier to implementing new problem domains has, based on student and teacher feedback, proven significant. Many students are able to conceptualise and design the problems and fitnesses they wish to implement, but lack the programming ability to implement them.

We also observed that students frequently had issues integrating concepts from different generative design systems towards a more complete understanding. In particular, students were often frustrated by generative design systems that did not provide immediate feedback on design experiments and confused by the number of "new" concepts introduced by evolutionary design systems. To address these issues we have used and extended a context free design grammar to provide a single environment to learn about design grammars, parametric design, and evolutionary design. The resulting system allows students to evolve parameterised context free design grammars. The remainder of this paper presents how we have used this system in the teaching evolutionary design by extending a design grammar, initially to introduce parametric design and ultimately to provide a platform for experimenting with interactive evolutionary design.

2 Description

The Context Free Design Grammar (CFDG) is a simple language written by Chris Coyne for generating 2D pictures, which supports the generation of complex images from simple programs. CFDG reads in a grammar file, executes its rules and renders an image file. Informally, programs in CFDG are sets of simple rules describing how to draw designs using a small set of primitive (terminal) shapes: circles, squares, and triangles. Rules may also call other rules and often call themselves recursively. When several rules share the same name this yields a non-deterministic grammar where each reference to the name results in one of the rules being chosen at random. An example of a CFDG program with an image it produces is given in Fig. **??**. Context Free[1] is a graphical development environment for writing CFDG programs and generating images and movies.

Students begin learning to implement generative design systems by experimenting with Context Free to develop CFDG programs. This initial experimentation allows students to explore the generative potential of recursive and non-deterministic grammars. By definition, CFDG[2] does not support variables. To extend the range of systems that students can explore within the Context Free environment we first developed an interpreter to replace parameters within CFDG files, allowing the space of parametric grammars to be explored.

[1] Context Free is open-source software and is available for download from `http://contextfreeart.org/`, it runs on Windows, Linux and Mac OS X operating systems.

[2] The inclusion of variables would violate the need to keep the CFDG rules free of context.

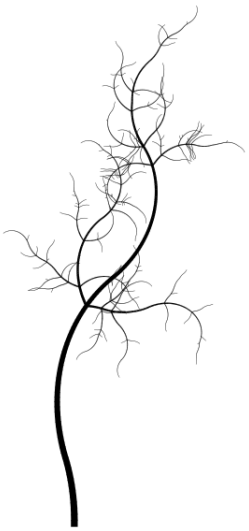
```
startshape SEED1

rule SEED1 {
    SQUARE{}
    SEED1 {y 1.0 s 0.99 r 1.5}
}

rule SEED1 0.05 {
    SQUARE{}
    SEED1 {y 1.0 s 0.99 r 1.5}
    SEED1 {y 1.0 s 0.6 r -60}
    SEED1 {y 1.0 s 0.5 r 60}
}

rule SEED1 0.05 {
    SEED1 { flip 90 }
}
```

(a) Source code



(b) Generate image

Fig. 1. An example Context Free Design Grammar and one of the images that it can generate. The image is defined by the variation code `MKY`.

2.1 Parametric Context Free

Parametric Context Free uses template files to define a set of parameters and a set of modified CFDG rules. A template file is a modified Context Free Design Grammar file that parameter definitions and uses those variables in place of constants in its grammar rules. Parameter definitions include a name, a minimum value and a maximum value. An example template is shown in Fig. **??**. This template defines twelve parameters and three rules. The first rule calls the second rule and then recursively calls itself with translation and rotation parameters. The second rule calls the third rule and then recursively calls itself with another translocation and rotation. The third rule draws a short line segment. Context Free automatically recurses until the shapes have dropped below a specific size threshold and then terminates.

The parameters are defined within the comments tags at the top of the file. Templates are designed so that when the parameters used within the rules are replaced with values, a valid CFDG grammar is generated. The definitions are placed within comments blocks, so that they can be left in the grammar files generated by the system. The parametric CFDG interpreter reads a template file and generates an instance by extracting the parameter definitions and substituting each reference to a parameter with a randomly generated value in the range allowed for the parameter. The values for each instance are written into

```
/* $R1 = [-1,1] */
/* $X1 = [-0.1,0.1] */
/* $Y1 = [-0.1,0.1] */
/* $R2 = [-1,1] */
/* $X2 = [-0.1,0.1] */
/* $Y2 = [-0.1,0.1] */
/* $R3 = [-360,360]*/
/* $X3 = [-1,1] */
/* $Y3 = [-1,1] */
/* $R4 = [-360,360] */
/* $X4 = [-1,1] */
/* $Y4 = [-1,1] */

startshape LINES2

rule LINES2 {
  LINES1 { r $R1 x $X1 y $Y1 }
  LINES2 { r $R2 x $X2 y $Y2 s 0.975 }
}

rule LINES1 {
  LINE { r $R3 x $X3 y $Y3 }
  LINES1 { r $R4 x $X4 y $Y4 s 0.975 }
}

rule LINE { TRIANGLE { s 0.025 1 } }
```

Fig. 2. A parameterised CFDG template with three rules and twelve variables.

CFDG files using the rules from the template so that they can be expressed into images. An example instance of the template from Fig. **??** is given in Fig. **??**.

```
startshape LINES2

rule LINES2 {
  LINES1 { r 0.6561 x -0.0447 y 0.0394 }
  LINES2 { r -0.4817 x -0.0756 y -0.0537 s 0.975 }
}

rule LINES1 {
  LINE { r -236.9877 x -0.7240 y -0.0138 }
  LINES1 { r -40.0891 x 0.4812 y -0.7348 s 0.975 }
}

rule LINE { TRIANGLE { s 0.025 1 } }
```

Fig. 3. A randomised instance of the parameterised rule from Fig. **??**.

Using the interpreter we developed an application, which allowed students to design and explore simple parametric design spaces using Context Free.

2.2 Evolutionary Context Free

The parametric CFDG system was further extended by developing an interactive genetic algorithm for students to evolve instances of CFDG templates. In the evolutionary CFDG, a parameter set in a template file defines the genotype for individuals and the rules are used to express genotypes into phenotypes. The evolutionary CFDG system uses a standard one-point crossover operator and per-gene mutation probabilities to generate children from parents.

An interactive evolutionary system was developed using the evolutionary CFDG system, where the phenotype (image) for each individual in a population is displayed side-by-side. The population shown in Fig. **??** were generated using the twelve-parameter grammar listed above. The space of possible designs generated from the three CFDG rules (comprising just seven lines of code) is very large, illustrating how quickly students could generate interesting design spaces to explore with the evolutionary CFDG system.

The interactive evolutionary CFDG system displays each population of image-based phenotypes to allow the user to select one or more favoured designs, which are used to generate the next population. A screen shot of the selection interface is shown in Fig. **??**.

The evolutionary CFDG application was developed in Processing [**?**], a programming environment, which the students were already familiar with. The code to the application was made available to the students, which allowed some to

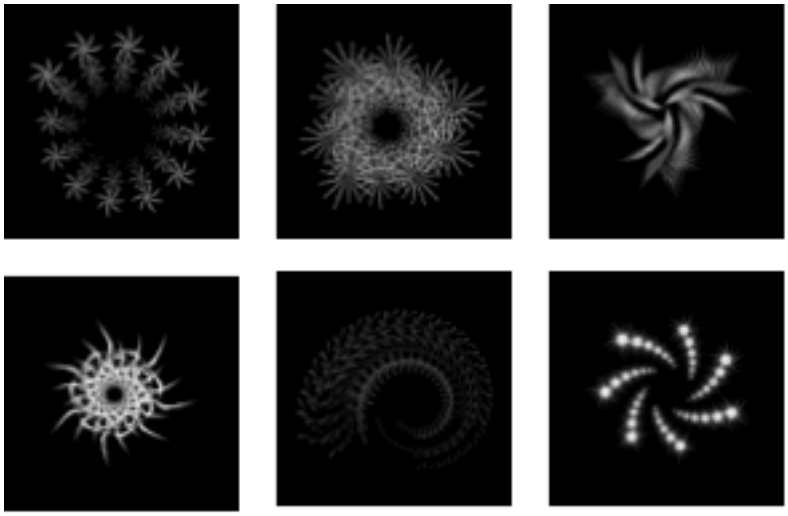


Fig. 4. Sample outputs from a template with two recursive rules and one drawing rule.

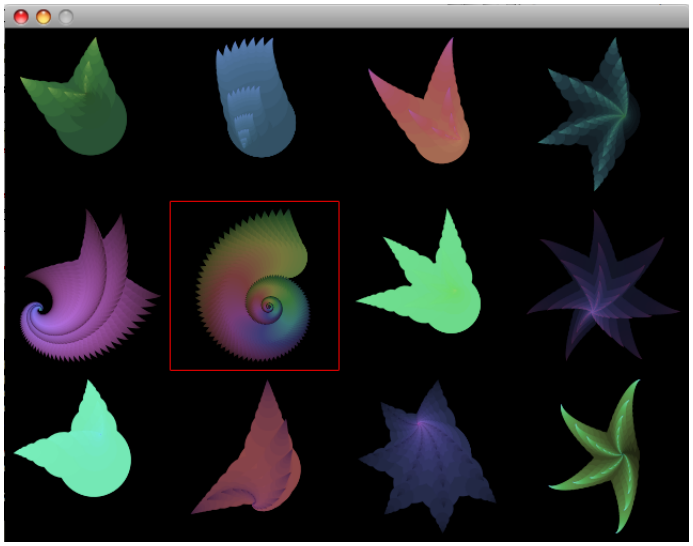


Fig. 5. The selection interface of the Interactive Grammar Evolver.

experiment with writing fitness functions to replace user selection. Some fitness functions such as surface area, X/Y symmetry, rotational symmetry, colour variance and brightness were provided as examples. Students created new fitness functions to guide the evolution of their parametric CFDGs. A screen shot from an application written by a student to evolve a circular grammar based on a fitness function that promotes brightness can be seen in Fig. **??**.

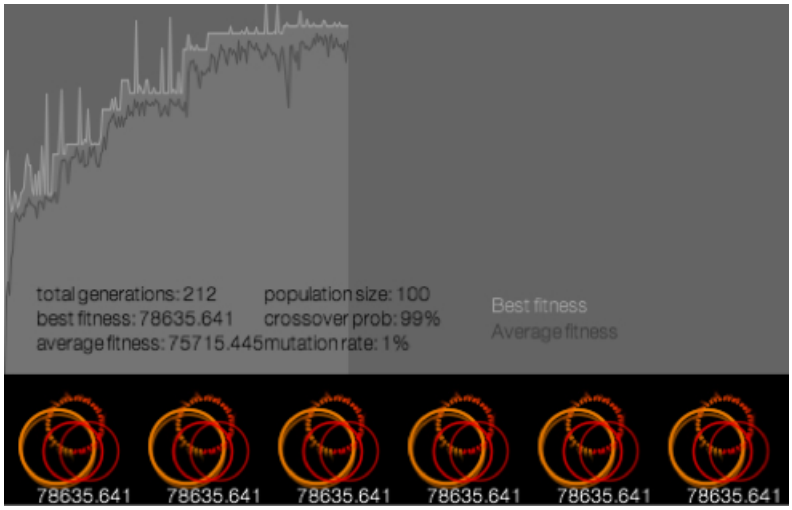


Fig. 6. The image-based fitness function mode, showing the fittest individuals in the population and a plot of fitness over time.

3 Results

Students were first taught design grammars, then parameterised grammars, then interactively evolved grammars and finally automatically evolved grammars. This progression gave the students time to learn to construct and explore rule-based design spaces before learning methods for searching them. Each step in the curriculum involved learning a new level of abstraction in the design of generative systems without presenting a large technical barrier to students. As a result, students who may have previously had great difficulties understanding the structure of a genetic algorithm were capable of adapting their design style to the medium of parametric rule-based systems. Examples of student designs can be seen in Fig. **??**, showing the breadth capable both within a student's generative system and between different students..

Students showed progress in being able to modify and understand image-based fitness functions. Students who were able to understand the existing fitnesses were able to explore the interaction between changing templates and

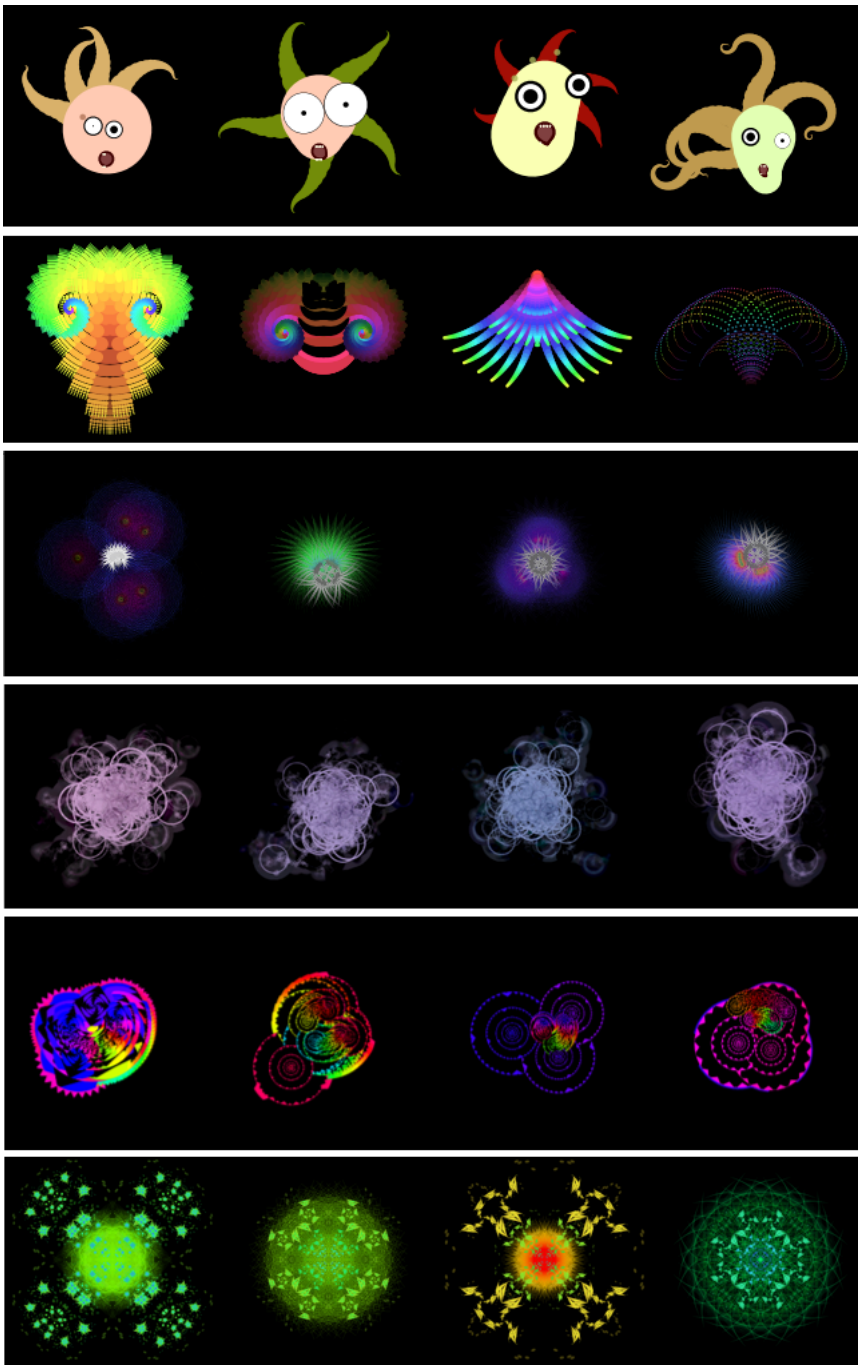


Fig. 7. A composite of grammars produced by different student-developed templates.

changing fitnesses, and those students who were able to create their own fitness functions also produced documentation showing they possessed a good understanding of how GAs work and can be applied. Students were able to communicate the concepts they wanted to evolve for - spikes, appendages, roundedness, gradients, pleasant colour combinations. There were more varying degrees of success when they came to develop those concepts into algorithms. Some students had difficulty following the implementations of the provided fitness functions (colour range, brightness, shape area, shape spikiness and rotational symmetry), but were able to apply and combine them.

Students responded positively to having the sections of code they had to deal with separated from those they didn't. The code was separated into three sections, the overall GA structure, the fitness functions, and the internal workings that were accessed by both sections but didn't need to be understood by students. This was, in verbal feedback from several students, a setup more conducive to learning than the code used for another assignment where the GA was simplified and presented complete.

4 Discussion

The interactive evolution CFDG system proved an effective way to teach design students the principles of design grammars, parametric design and evolutionary design. In contrast with previous teaching approaches, which involved students experimenting with configurable but fixed problems for each class of design system, the use of CFDG templates allowed students to implement their own problem domains very quickly.

The parametric template system, in which parameters can be inserted into any design grammar and interpreted as genes, allows students to experiment with altering design spaces and immediately see how they affect the search process. This allows students for whom implementing their own problem domain would be an obstacle to experiment with their own genotypes, their own genotype-to-phenotype expressions and (through the interactive evolutionary CFDG system) their own fitness functions. This allows students to gain a deeper understanding of how the design of problems, representations and fitness functions affects the use of genetic algorithms. Students were able to experiment with the development of evolutionary systems with relatively little programming. Compared to other methods of teaching evolutionary design systems we found this method to be highly effective and enjoyable for the students to learn and experiment with.

Future iterations of this teaching method would benefit from an increased focus on the interactive selection over the automated fitnesses. When teaching the automated fitness the applet would benefit from a way to select and combine fitnesses using the interface, rather than writing code. This would further lower the exposure of the students to the underlying implementation, which tended to unsettle them. The overall teaching methodology, however, showed great potential in the teaching of generative design systems to design students.

References

[1] Reas, C., Fry, B.: Processing: A Programming Handbook for Visual Designers and Artists. The MIT Press, 2007.