Full Presentation – A language-based approach to computational art

Shrutarshi Basu
Cornell University
basus@cs.cornell.edu

Chun Wai Liew
Lafayette College
liew@cs.lafayette.edu

November 14, 2011

Abstract

Language based tools have made their mark as popular computational art systems. However these tools generally use variants of popular programming languages like Python and Java. They require the user to become familiar with traditional imperative programming languages and understand concepts like functions, classes and objects. Thus they are more suited for programmers with artistic interests than for artists looking to leverage computational power. We present Metaphor – a language-based tool designed specifically for artists with little or no programming experience. Metaphor uses a declarative language to remove explicit control flow and expose clear relationships between the artists program-like descriptions and the images they generate.

1 The Problem

In recent years, language-driven tools such as Processing and Field have promoted the use of computers in graphics and art. Though they are powerful tools, their use of imperative programming languages require users to learn imperative syntax and concepts like functions and classes. Artists must also deal with problems such as syntax and type errors. Thus they distract the artists from the tools' actual purpose – enabling the creation of beautiful pieces of art.

Conversely, image manipulation and drawing programs such as Adobe PhotoShop, Adobe Illustrator and Inkscape allow artists direct manipulation of their images but do not allow them to create their own abstractions (apart from simple reusable components like layers). Though some such applications provide rich scripting interfaces, this requires the artist to deal with the problems of standard programming languages as described above.

2 A Proposed Solution

Metaphor takes a different approach to computational art. Instead of using a full-blown imperative language to describe graphical operations, the artist declares *relationships* between elements of the image. Metaphor provides a declarative language interface to an implementation of Lindenmayer systems – parallel rewriting systems that are capable of generating fractal and biological systems. The declarative language interfaces with graphical backends via a mechanism we call "contexts". This allows the backends to be written as Python classes but used as simple declarations from inside Metaphor.

For example, the following snippet in the Metaphor declarative language uses a small set of declarative rules to produce a complex, self-similar pattern after a number of recursive applications. The rendering context exposes a 2D drawing library via simple operations (forward drawing and turns):

```
System Sierpinski
    Axiom A

    Rule A => B right A right B
    Rule B => A left B left A

    Render A => forward(10)
    Render B => forward(10)
    Render right => right(60)
    Render left => left(60)
```

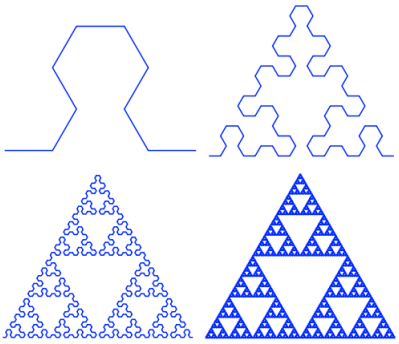

Figure 1: The Sierpinski Triangle

Thus Metaphor leverages declarative programming to provide two innovations over existing computational art tools – a declarative language in which to describe the images to be created and a mechanism to export functionality from graphics libraries to the declarative Metaphor language.

3 Preliminary Results

Metaphor has been used by introductory computer science students working with art students to create a variety of images exploring growth and symmetry. Feedback from the art students and their professors was used to create a number of specialized contexts. Metaphor allowed the artists to talk about their works and what they wanted to see in terms of their own metaphors without having to learn computational concepts and jargon. An exhibition at Lafayette College featured works created using Metaphor in print form on various media. Further work is currently under way to develop a graphical version of the Metaphor language using a block-based paradigm (similar to the Lego Mindstorms interface).