UNIVERSITY OF THE WEST INDIES
Department of Computing
COMP3652–Language Processors
Semester I, 2013
Lecturer: Dr. Daniel Coore

FRACTAL

Introduction

A fractal is a geometric object that is self-similar at all scales. One simple way to think of a fractal is to think of it being generated in discrete levels, each successively higher level being a better approximation to the actual fractal shape. A *generator* indicates how a level n fractal's approximation is generated from its level $n-1$ approximations, which are represented at a smaller scale. For the simplest types of fractals, a level 0 fractal is just a line segment. So the approximation at level 1 can often serve as a geometric representation of the generator, since we can usually interpret each line in the level 1 approximation as a segment along which the fractal should be drawn at a smaller scale. Figure 1 illustrates this process for a simple fractal, called the *Gosper* curve.

This view of the generator can usually inform us on how to define the generator. For example, the Gosper generator requires that two level $n-1$ fractal approximations are placed at right angles to each other so that the end points of the two fractals touch. They are oriented so that the other ends of the two fractal approximations lie along the line segment specified for the level n fractal to be drawn. To see this more clearly, observe how the rendered Gosper curve generator is a right isosceles triangle oriented so that its hypotenuse is aligned with the intended direction of the fractal. From that, we can deduce that the level $n-1$ fractals must each be drawn at the scale $\frac{1}{\sqrt{2}}$ relative to the level n fractal.

The language FRACTAL is a graphical language that enables its users to quickly generate fractals, and to combine them in interesting ways. It uses the *turtle graphics* system, first introduced in a programming language called LOGO, which was created by Seymour Papert in the 1970's, as a language that could be used to teach children programming. Although FRACTAL will use many primitive commands from LOGO, it is not a superset of LOGO.

In LOGO, there is a single turtle with a pen, which may be in either the up or down position. Whenever the pen is down, the turtle leaves a mark upon the screen, as it moves. The turtle always has a position on the screen, and a bearing. It can be given commands to move forward, or backward by a given distance, and to turn either left or right by a given angle. Each turtle command typically does only one of those actions.

For example, the Gosper curve could be defined by the following FRACTAL code:

```
def gosper fractal
   left(45)       // turn left by 45 degrees
   self(0.707)    // draw the fractal at 0.707 (1/√2) scale
   right(90)      // turn right 90 degrees
   self(0.707)    // draw the fractal at 0.707 (1/√2) scale
   left(45)       // point turtle in original direction
   end            // end of fractal definition body
```

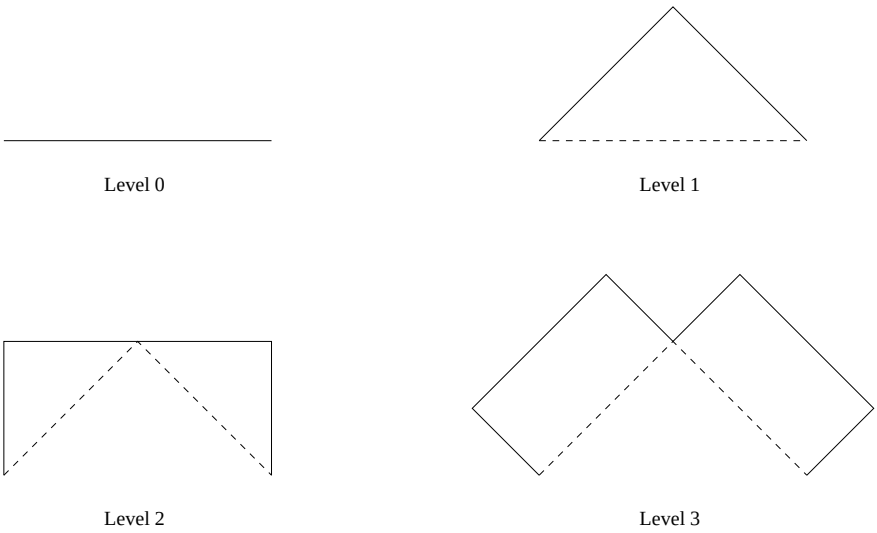


Figure 1: A level n fractal is made up of level $n-1$ fractals arranged according to the description provided by the generator. A level 0 fractal is a straight line. This example illustrates the Gosper "C" curve. It is created by placing at right angles, two copies of itself each scaled down by $\frac{1}{\sqrt{2}}$ (the level 1 fractal illustrates the generator).

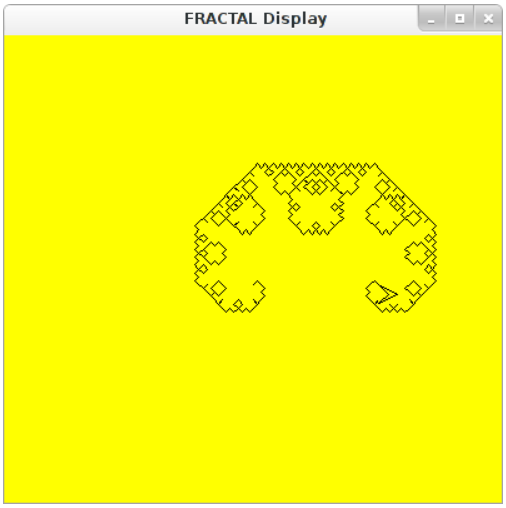


Figure 2: The Gosper curve rendered at level 8 and scale 100, which is evidently closer to its limiting shape.

To illustrate a fractal, we use the `render` command. For example, calling `render[8, 100] gosper` after defining `gosper` above would render the Gosper curve at level 8, along a horizontal line segment 100 units (turtle steps) long (see Figure 2). By default, the turtle starts at the centre of the display (coordinate (0, 0)) and the display is 400 turtle steps wide and 400 turtle steps high.

Primitive Statements and Expressions

A FRACTAL program is a sequence of statements, each of which typically causes some side effect, such as directly manipulating the turtle (by using a turtle command) or defining a new fractal generator.

Expressions in FRACTAL are either arithmetic, logical or fractal combinations, which denote, respectively, numerical, boolean or fractal values. Numerical values, in turn, can be either integer or real valued. Integers may always be supplied where reals are expected, but it is an error to pass a real where an integer was expected. Fractal values may be passed only where fractal values are expected.

Arithmetic expressions are combined using the usual operators (+, -, /, *) with the usual order of precedence. Logical operators include the comparators, >, <, =, as well as the boolean operators `and, or, not`. All the comparators have a lower precedence than any arithmetic operator, and `not` is higher than `and`, which in turn, is higher than `or`.

In the tables of special forms below, the variable n is used to denote an integer (arithmetic) expression, the variable r is used to denote a real valued expression, and the variable f is used to denote a fractal valued expression. All parentheses and brackets appearing in the tables are to be taken as part of the syntax of the statement or expression being documented.

Statements

The statements that directly manipulate the turtle, which have all been inherited from LOGO, are listed in Table 1.

Keyword	Abbrev	Meaning
`forward`(r)	`fd`	Move the turtle forward by the specified distance.
`back`(r)	`bk`	Move the turtle backward by the specified distance.
`left`(r)	`lt`	Turn the turtle left by the specified angle (in degrees), without changing its position.
`right`(r)	`rt`	Turn the turtle right by the specified angle, without changing its position.
`penup`()	`pu`	Raise the pen. Now, when the turtle moves, it will not draw a line.
`pendown`()	`pd`	Lower the pen. Now the turtle will draw a line as it moves.
`home`()		Restore the turtle to its initial position and orientation, without drawing a line.
`clear`()		Clear the turtle display.

Table 1: Table of FRACTALcommands that directly manipulate the turtle

Apart from turtle commands, FRACTAL contains commands that affect how fractals are created or rendered. They are listed in Table 2.

Fractal Valued Expressions

The primary way to denote a fractal is to define one using the `fractal` special form. A fractal's definition starts with the keyword `fractal` and ends with the keyword `end`. In between must be a non-empty sequence of statements, which are collectively called the *body* of the fractal. The body of a fractal is made up of statements that may occur at the top level, as well as one additional one called `self`.

The keyword `self` is valid only within a fractal generator's body and denotes the fractal generator that is being defined. It is by reference to the `self` special form that a fractal generator reproduces

Keyword	Abbrev	Meaning
`render` [n, r] f		Show fractal f approximated at level n, and scaled proportionately to a line segment of length r.
`render` f		Show fractal f approximated at the default level (9) and scale (100).
`level` n		Set the default level of subsequent calls to `render` to n
scale r		Set the default scale of subsequent calls to `render` to r
`save`		Save the current state of the turtle (on a stack).
`restore`		Restore a previously saved turtle state to be current.
`def` *name* f		Bind the given name to the fractal value f.
`self`(r)		Valid only within a fractal generator's body. Render this fractal at the scale r. Practically, this means that `self` draws the next lower level approximation of the current fractal at a scale r relative to the current scale.

Table 2: Statements other than turtle commands in FRACTAL

itself. It takes a real valued argument that indicates how much the smaller fractal should be scaled. The language FRACTAL provides two different combinators for fractal values. A fractal may be composed onto another to produce a single new fractal, or a fractal may be sequenced with another one. Table 3 summarises all of the special forms used to define and combine fractal valued expressions.

Keyword	Abbrev	Meaning
`fractal`		Declare the start of a fractal generator's body
`end`		Declare the end of a fractal generator's body
f_1 `@` f_2		Compose fractal f_1 with f_2
f_1 `!` f_2		Sequence fractal f_1 with f_2

Table 3: Commands for defining and combining fractals.

Means of Combination

When two fractals are composed, the second one is used wherever a reference to `self` occurs in the first, but a reference to `self` in the second is regarded as a reference to the overall combined fractal.

Sequencing two fractals is simply constructing a generator by starting the second where the first ends. Not all fractals have only one "end point" though: if a `restore` command is used, then the position of the turtle just before the restore is evaluated is regarded as an end point of the fractal. The rationale is that such a point would represent a "tip" of the fractal generator, as the

turtle is "picked up" and placed at a potentially non-local point to resume some previously deferred computation. The usual end of a fractal is always considered a tip. To illustrate the preceding discussion, below is an example of a tree fractal, which has two "tips":

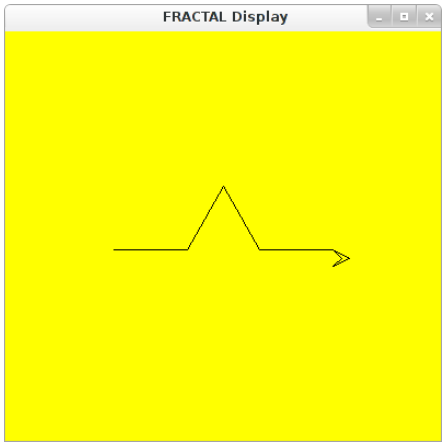
```
def tree fractal
  fd(1.0)
  save
  left(45) self(0.5)
  restore
  right(45) self(0.5)
  end
```

These means of combination should produce some interesting, and perhaps unexpected fractal shapes when they are rendered – probably because neither fractal operator is commutative. For example, while it might be easy to mentally visualise the generator resulting from composing `tree` with `gosper` (i.e. `tree @ gosper`) it is less obvious what `gosper @ tree` means, let alone what its fractal will look like.

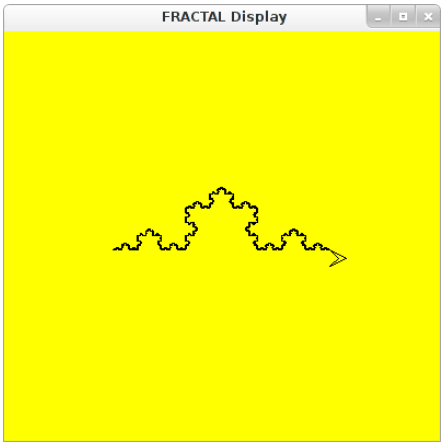
Examples

Below is the definition of the Koch curve, which also sometimes called the *snowflake curve* because when three Koch curves are used to replace the sides of an equilateral triangle, the resulting figure resembles a snowflake.

```
def koch fractal
    self(0.333)
    left(60) self(0.333)
    right(120) self(0.333)
    left(60) self(0.333)
    end
```



(a) The Koch generator resulting from calling `render[1, 200] koch` (after moving the turtle 100 steps back from its home position).



(b) The Koch curve resulting from calling `render[9, 200] koch` (after moving the turtle 100 steps back from its home position).