6.837 Introduction to Computer Graphics
Assignment 0: Iterated Function Systems (IFS)
Due Wednesday September 10, 2003 at 11:59pm

The goal of this assignment is to get familiar with C++ and with two simple libraries that we will use for linear algebra and images. The incidental goal is also to have fun with bizarre fractal objects: IFSs.

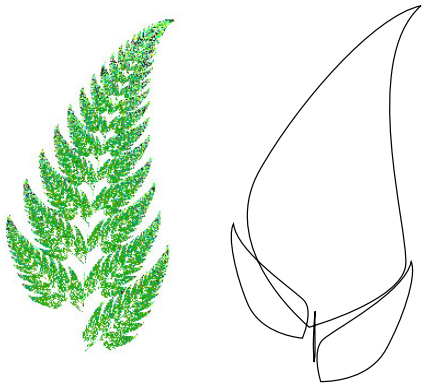

Figure 1: Barnsley's fern

IFS are self-similar fractals: a subpart of the object is similar to the whole. The classic example of an IFS is Barnsley's fern, where each subpart of the fern is exactly the same as the whole fern. IFS are described by a set of affine transformations (rotations, translations, scale, skew, etc.) These transformations capture the self-similarity of the object (see Figure 1). IFS can be defined in any dimension, but we will play with two-dimensional ones.

Formally, an IFS is defined by n affine transformations. Each transformation, f_i, must be contractive: The distance between points must be reduced. An *attractor* of the IFS is the object such that $A = \bigcup f_i(A)$. A is unchanged by the set of transformations: It is a fixed point.

We render an IFS by iterating the transform on random input points from the unit square. We approximate the fixed point by applying the transformation many times (see Figure 1). The algorithm is as follows:

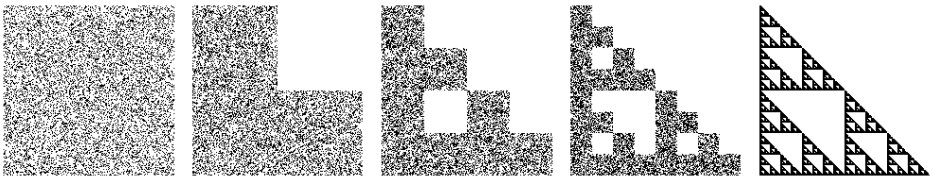

Figure 2: Sierpinski's triangle with 0,1,2,3 and lots of iterations

```
For a number of random points (x0, y0)
    For k=0 to big_number
        Pick a random i between 0 and n-1
        (xk+1, yk+1)=fi(xk, yk)
    Display a dot at (xk, yk)
```

To reduce the number of points necessary to make an image of reasonable quality, probabilities are assigned to each transformation, instead of choosing a transformation with uniform probability.

1 Tasks

- Write a C++ class `IFS` that renders iterated function systems, including the interface (in a file `ifs.h`) and the implementation (`ifs.C`). Your code should run under Linux or Windows. Specifications for the IFS class are below.
- Write the main program `main.C` that creates an `Image` instance, reads an IFS description from a file, renders the IFS to the image, and saves the image.
- Use the linear algebra library for the point and transformation representations.
- Perform proper memory management — free memory when an object is destroyed.
- Extra credit: create a new IFS, figure out the probabilities, change the color scheme, anti-aliasing, depth-first vs. breadth-first, etc. Include a short paragraph describing your extensions.

2 C++ class IFS

The IFS class should include:

- a field to store n, the number of transformations

- an array of matrices representing the n transformations
- an array of the corresponding probabilities for choosing a transformation
- a constructor that creates an IFS
- an input method that reads the IFS description
- a render method that takes as input an image instance, a number of points and a number of iterations
- a destructor that frees the memory of the various arrays (using `delete`)

3 Utilities

Images (`image.h`, `image.C`)

The `Image` class is used to initialize and edit the rgb values of images. Be careful — do not try to edit values outside the bounds of the image. The class also includes functions for loading and saving simple `.tga` image files. `.tga` files can be viewed with `xv` or opened in Photoshop and other Windows image viewers/editors.

Linear Algebra (`vectors.h`, `matrix.h`, `matrix.C`)

Linear algebra support for floating point vectors with 2, 3, and 4 elements (`Vec2f`, `Vec3f` and `Vec4f`) and 4x4 floating point matrices (`Matrix`). For this assignment, the `void Matrix::Transform(Vec2f &v)` function will be handy.

Parsing command line arguments & input files (`parse.C`)

Your program should take a number of command line arguments to specify the input file, number of points, number of iterations, output image size and output file. Make sure the following example works, as this is how we will test your program.

```
ifs -input fern.txt -points 10000 -iters 10 -size 100 -output fern.tga
```

The input data for an IFS is a file which contains n, the number of transforms, followed by the probability of choosing each transform and a 3x3 floating point matrix representation of the transform. Here's an example, from `sierpinski_triangle.txt`:

```
3
0.33
0.500000 0.000000 0.000000
0.000000 0.500000 0.000000
0.000000 0.000000 1.000000
0.33
```

```
0.500000 0.000000 0.500000
0.000000 0.500000 0.000000
0.000000 0.000000 1.000000
0.34
0.500000 0.000000 0.000000
0.000000 0.500000 0.500000
0.000000 0.000000 1.000000
```

Sample code to parse input files and command line arguments is given in `parse.C`.

Other

Random numbers can be obtained using the `drand48()` or `rand()` and `RAND_MAX`. See `<stdlib.h>`.

A simple `Makefile` for use with `g++` and linux is provided.

4 Hints

- To debug your code, set the number of iterations to one. This will allow you to check that you got the transformations right.
- Be careful, arrays are indexed from 0 to n-1 in C++. Reading beyond the bounds of the array will probably result in a segmentation fault.
- Use `assert()` to check function pre-conditions, array indices, etc. See `<assert.h>`.

5 Additional references

M.Barnsley, Fractals Everywhere, Academic Press, 1988.

`http://spanky.triumf.ca/www/fractal-info/ifs-type.htm`

`http://www.cut-the-knot.org/ctk/ifs.shtml`