

Graphics Package Application: Fractals

COMP1100 — Introduction to Programming and Algorithms

Clem Baker-Finch

Australian National University
Semester 1, 2006

A cool application of the graphics package, and an interesting case of general recursion.

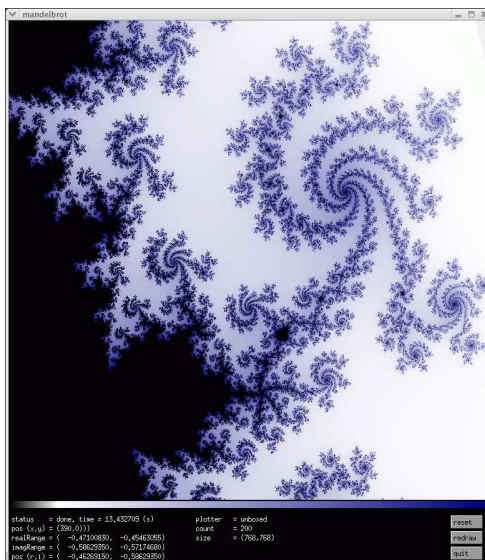
Fractals

Fractals are *naturally recursive shapes* — they are defined by rules that specify how to make the shape from *smaller* and **simpler** copies of itself.

Fractals are widely used in computer animation (games, movies, . . .)

For example, see www.speedtree.com

Convincing images of clouds, topography, trees, swarms etc. can be created using fractals.



A Simple Tree Fractal

An elementary tree can be a trapezium:

```
branch :: Picture
branch = [Polygon [(30,0), (15,300), (-15,300), (-30,0)]]
```

Which just looks like a rudimentary trunk.

(Stump.hs or trunk in Tree.hs)



Notice that this is constructed from several smaller versions of the original shape — the essential idea of a fractal.

Putting it together:

```
limbs :: Picture
limbs =
  trunk ++
  [Transform [Translate 0 300]      limb] ++
  [Transform [Translate 0 240, Rotate 20]  limb] ++
  [Transform [Translate 0 180, Rotate (-20)] limb] ++
  [Transform [Translate 0 120, Rotate 40]   limb] ++
  [Transform [Translate 0 60, Rotate (-40)] limb]
  where limb = [Transform [Scale 0.5 0.5] trunk]
```

We can make a more interesting tree by attaching limbs to the trunk.

Each limb is just a smaller version of the trunk:

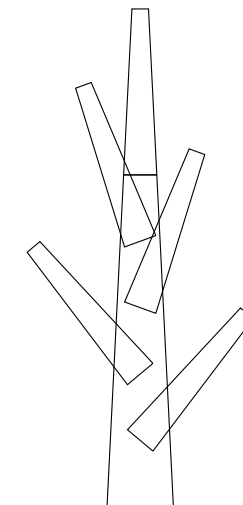
```
where limb = [Transform [Scale 0.5 0.5] trunk]
```

We put one limb on top of the trunk:

```
trunk ++
[Transform [Translate 0 300] limb]
```

and 4 others sticking out the sides of the trunk:

```
[Transform [Translate 0 240, Rotate 20]   limb] ++
[Transform [Translate 0 180, Rotate (-20)] limb] ++
[Transform [Translate 0 120, Rotate 40]   limb] ++
[Transform [Translate 0 60, Rotate (-40)] limb]
```



Carrying on ...

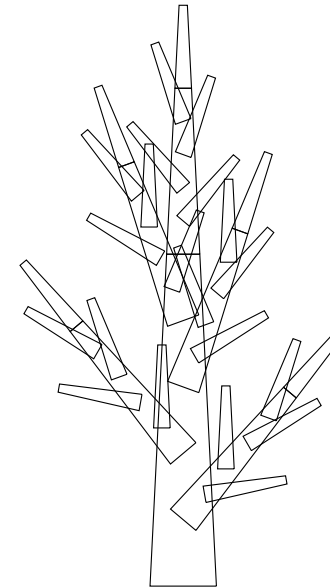
An even more tree-like shape ...

Key idea:

Instead of small versions of `trunk`, attach small versions of `limbs` to the trunk, *using the same rules*.

The definition of the `tree` picture is identical to `limbs`, except that we use a half-size version of `limbs` instead of `trunk`:

```
where branch = [Transform [Scale 0.5 0.5] limbs]
```



Putting it together again

```
tree :: Picture
tree =
  trunk ++
  [Transform [Translate 0 300]      branch] ++
  [Transform [Translate 0 240, Rotate 20]  branch] ++
  [Transform [Translate 0 180, Rotate (-20)] branch] ++
  [Transform [Translate 0 120, Rotate 40]   branch] ++
  [Transform [Translate 0 60, Rotate (-40)] branch]
  where branch = [Transform [Scale 0.5 0.5] limbs]
```

Generalising

We could make more and more interesting pictures by continuing this process. For example, the next step would be to add half-size `trees` to the trunk in the same arrangement, and so on.

What is the process? Can we generalise?

The `trunk` image is a fractal of *degree 0*

The `limbs` image is a fractal of *degree 1*

The `tree` image is a fractal of *degree 2*

How would we make a fractal of *degree 3*?

degree 4? degree 5?...

Generalisation Idea:

Make the *degree* a parameter to the tree fractal function:

```
tree :: Int -> Picture
```

`tree 0` will be the same as `trunk`.

`tree 1` will be the same as `limbs`.

`tree 2` will be the same as `tree`.

`tree 3` will be more complex, and so on.

Playing around...

- Colour shading adds to the effect ([TreeColour.hs](#))
- Making small changes to the parameters (limb locations and rotations) gives a different looking tree ([TreeColour2.hs](#))
- There are other simple fractals: Koch (Snowflake) curve, Sierpinski triangle, Dragon Curve, etc. See Week 7 lab exercises. ([Snowflake.hs](#), [Sierpinski.hs](#))

Putting it together:

```
tree :: Int -> Picture
tree 0 = trunk
tree n =
  trunk ++
  [Transform [Translate 0 300]      prev] ++
  [Transform [Translate 0 240, Rotate 20] prev] ++
  [Transform [Translate 0 180, Rotate (-20)] prev] ++
  [Transform [Translate 0 120, Rotate 40] prev] ++
  [Transform [Translate 0 60, Rotate (-40)] prev]
  where prev = [Transform [Scale 0.5 0.5] (tree (n-1))]
--
```



greyTree 5