Visual Examples of Recursion

Ben Stephenson

Department of Computer Science

University of Calgary

ben.stephenson@ucalgary.ca

Abstract

Because recursion is generally introduced early in the curriculum, the range of problems that can be used to motivate its study is limited. We describe three interesting visual problems that use recursion effectively. Each problem demonstrates the utility of recursion in an engaging way while being appropriate for students nearing the end of CS1.

1 Motivation

Solving problems recursively requires students to expand the way that they look at problems. Students often resist this change, in part, because they see few practical reasons for using recursion. We believe that this is partly a reflection of the traditional examples that we have used to teach recursion. As such, we have revised our course materials to include examples designed to meet three goals:

Utility of Recursion: The problem being solved must be at least as easy to solve using recursion as it is to solve using imperative control structures.

Relevance: The problem under consideration should serve some practical purpose.

Engaging: The problem being solved should be interesting to the students who are being taught.

2 Traditional Examples

Our traditional examples failed to meet some, or all, of the goals listed previously. In our experience, the examples we used often left students with the mistaken impression that recursion isn't very useful, or that it is too inefficient to solve interesting problems.

2.1 Factorial

Good:

- Simple recursive definition
- Students are already familiar with the problem

Bad:

- Using a loop is just as easy (perhaps easier)
- Problem is not very interesting

2.2 Fibonacci Numbers

Good:

- Recursive solution is simpler than the iterative solution
- Demonstrates the concept of multiple base cases

Bad:

- Recursive solution is too inefficient for even modest values
- Who wants to compute Fibonacci numbers?

2.3 Tower's of Hanoi

Good:

- Can be demonstrated with physical props / animation

Bad:

- Students recognize that the problem is contrived

3 Fractals

Fractal images are self-similar, meaning that each part of the image is a smaller version of the whole. As such, fractal images often have relatively simple recursive definitions.

3.1 Fractal T-Square

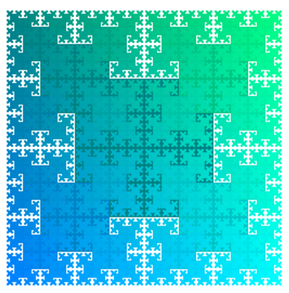
The Fractal T-Square is constructed by repeatedly drawing a square in the center of a square region. It can be generated using about a dozen lines of Python code.

```
def tsquare(x,y,w,h):
  if (w < 4) or (h < 4): return

  print "color",0,(x+256)/2,(y+256)/2
  print "fillrect", x + w/4.0, y+h/4.0, \
                    w/2.0-1, h/2.0-1

  tsquare(x,y,w/2.0,h/2.0)
  tsquare(x+w/2.0,y,w/2.0,h/2.0)
  tsquare(x,y+h/2.0,w/2.0,h/2.0)
  tsquare(x+w/2.0,y+h/2.0,w/2,h/2.0)

print "color 255 255 255"
print "clear"
tsquare(0,0,256,256)
```



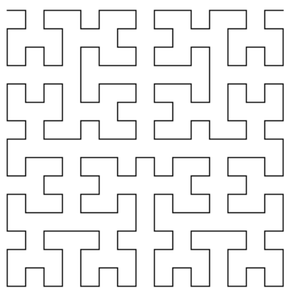
3.2 Hilbert Curve

The Hilbert Curve is defined by a pair of mutually recursive equations:

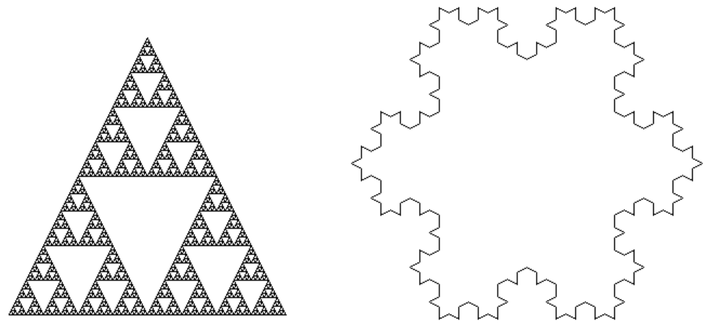
```
R(0)      = do nothing
R(k > 0) = - L(k-1) F + R(k-1) F R(k-1) + F L(k-1) -

L(0)      = do nothing
L(k > 0) = + R(k-1) F - L(k-1) F L(k-1) - F R(k-1) +
```

Within these definitions, '+' denotes a 90 degree left turn, '-' denotes a 90 degree right turn, and F denotes drawing a forward line segment one unit in length. Using a turtle graphics package makes it relatively easy to represent these equations as a pair of recursive functions which draw the Hilbert Curve.



3.3 Other Simple Fractals



3.4 Fractals Summary

Utility of Recursion: Drawing fractals iteratively requires a stack or a queue. Our students are not introduced to these data structures in CS1.

Relevance: While simple fractals have limited value as art, beautiful fractal artwork has been created. In addition, fractals can be used for practical purposes such as artificial terrain generation.

Engaging: While fractals are quite mathematical in nature, most students seemed to enjoy working with them. We believe this was partially because of their visual nature. In addition, fractals were something that some students had heard of previously but had never had the opportunity to study.

4 Floodfill

The floodfill algorithm is used to change a region of arbitrary shape within an image from one color to another.

Basic Algorithm:

- For the pixel at location (x,y)
  - If the pixel is currently the color that is to be changed
    - Use recursion to change pixels starting from one pixel above the current pixel
    - Use three additional recursive calls to change pixels starting one pixel left, one pixel right and one pixel below the current pixel

Most students found the floodfill algorithm more difficult to implement than drawing the Fractal T-Square. While drawing the fractal provided students with visual feedback each time a recursive call was made, they did not receive the same kind of feedback when implementing floodfill because they were modifying an image in memory rather than drawing directly to the screen.

Our students were engaged by this problem because it allowed them to implement a tool that they had used in various paint programs. Several students expressed interest in optimizing the algorithm, further suggesting that they found this problem relevant and engaging.

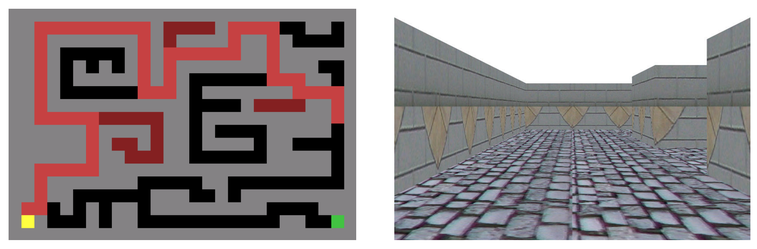
5 Maze

Two dimensional arrays / lists are an intuitive representation for a maze. Within the list, each element can represent:

- Filled regions which cannot be passed through
- Open spaces where a person can walk

In addition, one element is marked as the entrance, and another is marked as the exit.

Students wrote a program to find a path through a maze as their final assignment in CS1. They represented filled and open areas using different colored squares. Then they updated the display to show that progress was being made toward the solution. Their animation marked squares that were along the path from the entrance to the exit, and squares that were visited but were not ultimately part of the solution.



Students were given the opportunity to animate the solution in three dimensions. Even with a high level library, this added significant complexity without reinforcing the concept of recursion. Only one student completed this optional challenge.

6 Conclusion

Using visual examples while teaching recursion helped us effectively demonstrate the utility and relevance of recursion in an engaging manner. Based on our student's behavior, we strongly believe that we were successful in motivating students to study recursion while also successfully addressing misconceptions such as "anything done recursively is just as easily done with a loop" and "recursion is too slow to be practical for *real* problems".