Comp1130 Programming As Problem Solving(Advanced)

Assignment 2 Technical Report

Fri May 5,2018

Tutorial Group:Tuesday 2-4pm

Tutor:Debashish Chakraborty

University ID:u6683369

ANU College of Engineering & Computer Science

Introduction

In this assignment, I write code for how to draw shapes by calculating the integer points that the shape should have. This time I would like to get my extension mark both on smoothing graphs and conway's game of life.

Rectangle

In rectangle part, there are 4 ways of writing the code, but the idea is similar. You can use 4 recursions, 4 foldleft, 2 zips or 4 list comprehension to get the same correct shape. As rectangle has 4 sides, so we need to output all integer points that are on those sides. Because we know that [1..10] means a list of integers that are from 1 to 10. Then we can use [min a c .. max a c] to indicate a list of integers from a to c, and as each time either x or y coordinate of the point is the same, we can know all the points on this side.

Line

In line part, by looking up on the internet, first we learn that how to draw a positive gradient line in the canvas. And hence by changing the sign of x and y coordinates or swap x and y coordinates, we can then get the line in all quadrants. The bresenham's line algorithm simply introduce an algorithm that tell us where to draw the point on the line by accumulating error value. The only thing that I find difficult in this part, and also along this whole assignment is that it's difficult to build a loop in Haskell, but after I found out that there's a built-in function called iterate, which is call the function over and over again, we are supposed to do that, in this case, I put a zip in the front which means that call step again and again until I have zip all the tuples(points).

Also, there's a built-in function called nub, its function is to take all the points which occur more than once out and reduce its occurrence to once.
(Ex:nub[1,2,2,3,4,4,]=[1,2,3,4]) The nub function is used all the way through my assignment 2, its main purpose is to get me pass the doctest as it doesn't allow me to have same points.

Polygon

In polygon part, our task is to connect lines together, so we need to call lineRaster function in this part. The main feature that I would like to talk about in this part is that I used foldleft along with anonymous function(\...), in the polylineRaster case, it means that I take all the elements from side and apply them to lineRaster function, and put all the results in a list. The reason why I am using anonymous function here is to save the lines in the assignment, because otherwise I need to write where to define my function or write function somewhere.

Circle

In circle part, our task is to draw circle using mid-point circle drawing algorithm. To do that, we need to split it into 8 parts, because the circle has a degree of 360, if we cut it into 8 parts, each parts have a degree of 45. That means as x increase by 1, y also increase by 1(according to the quadrants). Also to generalize all cases in 8 parts, I construct a determine variable:d similar to line one, to determine how much x and y should vary. The only thing difficult in the circle part is the type in radius, we need to specify every single value's type, and in circle case, we use fromIntegral.

Test for first part

I test my program's correctness and efficiency by doctest, it shows that the first part works perfectly, also compiles with no errors and warnings.

Extension: Smoothing shapes

In the smoothing shape part, I would like to talk about the line part first. For the line, we have 3 parts: the direction that the line should be drawn(x-axis direction or y-axis direction?), first end point and second end point's handlement(we can't generalize these two situations), remain points' handlement.

Different from bresenham's line algorithm(handling integers all the time),in Wu's algorithm, it does involves floating value's calculation which will affect the efficiency of running the program. The thing that I can improve next time is to think of a way to improve its efficiency. We need to define its gradient first so we can use it throughout the whole algorithm, our task is to distribute the gradient to every single point. First we have a initial value, which we can see in remainpoints function,(intery = yend + gradient), and in the loop(iterate), it plus gradient every time, so the coordinates are evenly distributed along with its colour. The sum of all the surrounding shade of a pixel is 1, and its darkness is according to the distance to the pixel, in this way we can create a smooth effect.

Ipart function is used to calculate the integer part of a floating value, fpart function is used to calculate the decimal part of a floating value, rfpart function is used to calculate 1 – the decimal part of a floating value. We are using these functions to get the intery y's decimal part and 1-decimal part to determine the shadiness of the top and bottom points.

The steep is the value that determine the direction that the line should be drawn, everytime it occurs, for me to plot all the situations, if it's steep, then I swap the coordinates x and y, and otherwise I keep the same. (x0'>x1' does the similar thing with steep)

For the remainpoints function, we get rid of first and second end points in line174 as these two are special cases. In singlepoint function, we plot the previous and next integer point of inty and its correspondent shadiness.

For the circle part of smoothing, it's rather simple. We just need to consider when the circle is in first quadrant, so the initial value of x is r and initial value of y is 0. The idea for circle is that as x decreases, y increases.(As it's in first quadrant)

The dc function is to get the x value according to y value, and then get the decimal part of x then similar to line part, we can then plot the top and bottom points of it and its shadiness.

Then it's time to build a loop for circle, first we use let..in to define initial value and their types. We iterate the function until it reach the point that cx > cy, in each step cy need to plus 1, then for cx we need to determine if the last x point's decimal part is larger than this time, if that's true then cx-1 otherwise cx keep the same. Finally similar to midpoint circle algorithm, we can then draw all 8 parts of circle.

There's one similarity in Xiaolin Wu's algorithm and Bresenham's algorithm, we all have an initial value, we have a direction that either x or y coordinates is continuously plus 1 or minus 1, the another direction is according to some judgement(whether it's steep or whether x0>x1..) to determine whether the x or y value should be plus 1 or minus 1, each time that the x or y coordinates are plus 1 or minus 1, we can then generate one point or two points. Actually I did think that I can generalize all the shapes and algorithm into one single function and then add more judgement function to make it works. But now I don't think I am competent to do that.

Extension: Conway's game of life

An rather simple introduction for the Conway's game of life is that if a cell survives in last generation and have two other cells alive around it then in the next generation it will still be alive. However, if a cell have three other cells alive around it, then in next generation it will also be alive. In any other situations, the cell will be dead.

At start, I have tried to calculate all the situations that it could occur and write them all in. But by doing that, it makes my code rather complicated and very slow to run which is not going to make our criteria(0.5second refresh) as the calculation will be really big. Then I think that I should just calculate the situation that how many cells are still alive, I did this by recording how many cells are being marked three times, which means that in last generation it has three cells alive around it, also same for the cells being marked twice, in these two situations, a cell will be alive.

To get all the neighbor point, I use group and sort, first sort them and then group them together so I can tell how many time a cell is marked by seeing its length. Then I filter

all points that has a length of two or three, which is the new generation's point that we need to output as.

Also, we can't let the cells out of the boundaries, so in line25-29, I restrict them so they will not escape the place that they are supposed to be.

Next I am going to talk about how it operates and how the handletime works. First, if you go to Model.hs, you can see that I have added animation to the model type, that's because when I change flip const to 0.5*flip const, for some reason it doesn't work. So aparting from learning monad to change the time in Haskell, I rather store a value of time in the type of animation. According to my test on the handletime, no matter what it will refresh every 0.05 seconds. In handletime, I am asking if the game is running or not and if the time pass 0.5 seconds,(but actually I think 0.4 seconds is more appropriate) if the situation satisfies one of them, then we need to refresh.

According to the criteria, I need to add a key "G" to start the game or end the game. (line 66-73 in controller.hs) If the game is not running, and there's some shape on the canvas(not empty) then after G is pressed, the game should start. Otherwise end the game.

Finally, getinitAnimationState is to turn the shape into many single pixels so we can use these pixels to calculate things easier. And I change the updateView function in View.hs to patternmatch against normal function and conway's game of life.

Conclusion

Assignment 2 is harder than assignment 1,(for assignment 1 I use patternmatching and case all the time while in assignment 2 I am competent to play around with list and use foldleft, lambda expression, iterate…) however it's joyful to write code and solve problems. I have learnt so many stuff, the one thing that I think it's interesting is the (function <$>) actually have the same meaning as (map function $), so I change all my map to this little thing. By learning all these new stuff, I feel I am more competent and prepared to coding.

References

Xiaolin Wu's line algorithm, Wikipedia, https://en.wikipedia.org/wiki/Xiaolin_Wu%27s_line_algorithm

J.E. Bresenham, Algorithm for computer control of a digital plotter, IBM Systems Journal,
http://dx.doi.org/10.1147/sj.41.0025

M.L.V Pitteway, Algorithm for Drawing Ellipses or Hyperbolae with a Digital Plotter,

The Computer Journal 10(3):282–289, January 1967,
http://dx.doi.org/10.1093/comjnl/10.3.282

Xiaolin Wu, An Efficient Antialiasing Technique, Proceedings of the 18th Annual Conference on Computer Graphics, ACM SIGGRAPH Computer Graphics 24(4):143–152, July 1991,
http://dx.doi.org/10.1145/127719.122734

Anti-aliased Line | Xiaolin Wu's algorithm, Geeksforgeeks,
https://www.geeksforgeeks.org/anti-aliased-line-xiaolin-wus-algorithm/

Bresenham's line algorithm, Wikipedia,
https://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm

Conway's game of life, Wikipedia,
https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life