Design for COL216 Assignment 1

Shrey J. Patel,
2019CS10400

Aayush Goyal
2019CS10452

February 2021

1 Aim

To design an assembly code for calculating the algebraic area under the curve with respect to X-axis formed by joining integer co-ordinates, given as input, by straight lines.

2 I/O specifications

Input:

1. An integer n, equal to the number of input points. n should be greater than 0.
2. A sequence of integers(2*n) corresponding to the co-ordinates, in the order x,y in sorted order in terms of x-coordinate.

Output: A floating point(float) number corresponding to the area.

3 Approach

3.1 Basic Idea

We noted that, because the curve is made by joining finite number of integer co-ordinates by straight lines, the enclosed area is made of polygons, and precisely only triangles, rectangles or trapeziums. So, the net area is the sum of areas of these polygons.

And because the points are given in order of increasing x-coordinates, we take two adjacent coordinates(in terms of x), and calculate areas between them and the X-axis and finally add up areas for all such pairs of adjacent coordinates to obtain the net area.

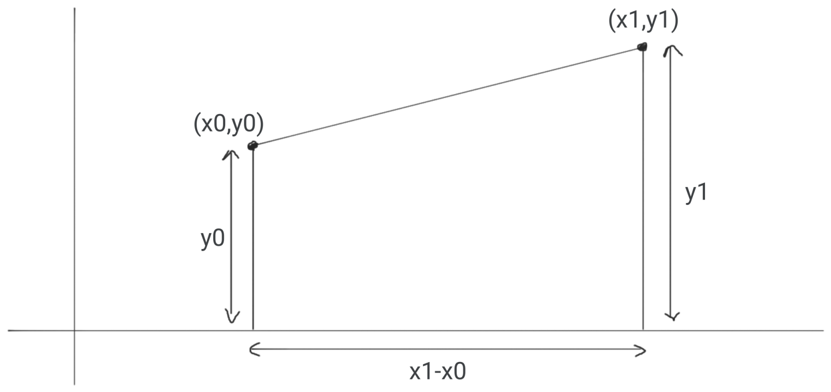
3.2 Calculating individual areas

Considering an adjacent pair of coordinates as (x_0, y_0) and (x_1, y_1) such that $x0 \leq x1$, the individual area depends on two cases:

3.2.1 Case 1: y-coordinates are of same sign

The enclosed area will be a trapezium, whose area is:

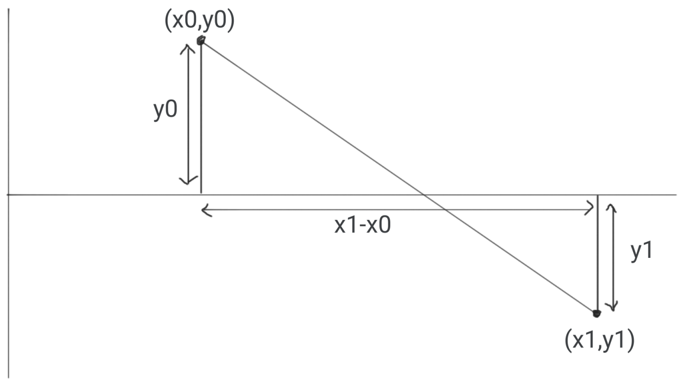
$$(x_1 - x_0) * (y_1 + y_0)/2$$



3.2.2 Case 2: y-coordinates are of different sign

The enclosed area(algebraic) will be the subtraction of the areas of triangles formed by X-axis as transversal, which also turns out to be:

$$(x_1 - x_0) * (y_1 + y_0)/2$$



Note that any of these areas can be negative, depending on y coordinates

4 Code Design and Implementation in MIPS assembly language

Included in code.asm

4.1 Algorithm

Using the above approach, we designed an iterative algorithm which iterates over all pairs of adjacent coordinates, and in each iteration, the area calculated as above is accumulated in a variable *sum*.

Area = $\sum_{i=1}^{n-1}(x_{i+1} - x_i) * (y_{i+1} + y_i)/2$

4.2 Analysis

n = No. of points, b = Max No. of bits in co-ordinates

1) **Space Complexity**: $O(b)$
We have used a constant number of registers (four for current pair of coordinates i.e. 2 for each coordinate, one floating point register for the current sum, etc.) and each register will store at max b bits, so total space complexity is $O(b)$.

2) **Time Complexity**: $O(nb^2)$
There are n points and so n-1 iterations of the loop. And in each iteration, addition and subtraction take $O(b)$ time and multiplication takes $O(b^2)$ time, so total time taken across all iterations = $O(n * b^2)$

Note that in code.asm, we have done the conversion of integer to floating point number using division(which consumes $O(b^2)$ time) only after the completion of the loop instead of converting int to float in every iteration, thus trading off some constant space for making the time of computation more efficient.

4.3 Testing Strategy

We made a random test case generator, *tc_generator.py*, which generates a test-case with random number(between 5 to 10) of random co-ordinates. We then determined the correct answer by calculating the same area generated in a *gold.py* file which implements the same algorithm. We then used the same test-case in QtSpim to generate the area in code.asm file and matched the outputs generated in both these files.

We have randomly generated three types of test cases:

1. When all the values of y are positive, that is the whole curve thus formed lies above the x-axis. These testcases are present in the TestCases/y_positive directory. The net area calculated will have a positive value.

2. When all the values of y are negative, that is the whole curve thus formed lies below the x-axis. These testcases are present in the TestCases/y_negative directory. The net area calculated will have a negative value.
3. When the values of y can be randomly positive or negative. In this case some part of the graph lies below the x-axis and some lies above. The convention we have followed is to consider area with sign (as one does during normal integration to consider the area below x-axis to be negative). The testcases are present in TestCases/y_mixed directory. Here the area can either positive or negative.

The correct (gold) outputs of these testcases were calculated using python code which implements the same function. All the code.asm outputs matched with the gold outputs. Apart from these we made some manual testcases in which we checked whether it is throwing exception for the case when the given value of n is less than 1 and it was doing this correctly.
Hence all the test cases have helped us to ensure that our code.asm is correct. Getting correct outputs on some test cases can never be the right criteria to judge if a Algorithm is correct or not but extensive checking can still ensure us that we have done the things correctly.

Note: The value of input coordinates should be such that they can be expressed as 32 bit signed integers and the net final area that comes can also be represented as a 32 bit signed integers, otherwise there will be overflows and the results will not be correct.