Paul Johnston

Convex Hull report

Sec 001

1. Source code

#Tangent line class with index of left point in left array and index of right point in right array

class Tan:

def \_\_init\_\_(self, indexL, indexR, slope):

self.indexL = indexL

self.indexR = indexR

self.slope = slope

#Gets slope of two points

def slope(self, p1, p2):

rise = p1.y() - p2.y()

run = p1.x() - p2.x()

return rise/run

#Returns the hull and the index of right most point

def clockwiseOrder(self, points):

if len(points) == 1:

return points, 0

elif len(points) == 2:

return points, 1

else:

clockwisePoints = [points[0]]

slopeTo1 = self.slope(points[0], points[1])

slopeTo2 = self.slope(points[0], points[2])

#In the rare case the slopes are the same we see which point is closer x-wise

if slopeTo1 == slopeTo2:

if points[1].x() < points[2].x():

clockwisePoints.append(points[1])

clockwisePoints.append(points[2])

return clockwisePoints, 2

else:

clockwisePoints.append(points[2])

clockwisePoints.append(points[1])

return clockwisePoints, 1

#If the slopes aren't equal, the point with the highest slope is next in clockwise ordering

if slopeTo1 > slopeTo2:

clockwisePoints.append(points[1])

clockwisePoints.append(points[2])

else:

clockwisePoints.append(points[2])

clockwisePoints.append(points[1])

#Which of those points is right most?

if clockwisePoints[1].x() > clockwisePoints[2].x():

return clockwisePoints, 1

return clockwisePoints,2

#Combines the two hulls

def combine(self, left, rmiL, right, rmiR):

#This is the upper tan to start with, right most point of left hull and leftmost point of right hull

#I'm using indexes to keep track of points in order to go counter clockwise or clockwise

upperTan = Tan(rmiL, 0, self.slope(left[rmiL], right[0]))

#self.showTan(left[upperTan.indexL],right[upperTan.indexR])

moved1 = True

moved2 = True

#While either side of the tangent has moved, we will continue to search for the upper tangent

while moved1 or moved2:

#We move the left point counterclockwise, if the new slope is less than the old slope, update the slope

if self.slope(left[(upperTan.indexL - 1) % len(left)],right[upperTan.indexR]) < upperTan.slope:

#self.eraseTan(left[upperTan.indexL],right[upperTan.indexR])

upperTan.indexL = (upperTan.indexL - 1) % len(left)

upperTan.slope = self.slope(left[upperTan.indexL],right[upperTan.indexR])

#self.showTan(left[upperTan.indexL],right[upperTan.indexR])

moved1 = True

#If the new slope is greater than or equal to the old slope, do nothing

else:

moved1 = False

#We move the right point clockwise, if the new slope is greater than the old slope, update the slope

if self.slope(left[upperTan.indexL],right[(upperTan.indexR+1) % len(right)]) > upperTan.slope:

#self.eraseTan(left[upperTan.indexL],right[upperTan.indexR])

upperTan.indexR = ((upperTan.indexR + 1) % len(right))

upperTan.slope = self.slope(left[upperTan.indexL],right[upperTan.indexR])

#self.showTan(left[upperTan.indexL],right[upperTan.indexR])

moved2 = True

#New slope was less than or equal than old slope, do nothing

else:

moved2 = False

#Compute the lower tangent

lowerTan = Tan(rmiL, 0, self.slope(left[rmiL], right[0]))

#self.showTan(left[lowerTan.indexL],right[lowerTan.indexR])

moved1 = True

moved2 = True

#We keep trying to find a better lower tangent if either of the points of lowerTan has changed

while moved1 or moved2:

#We move the left point of lowerTan clockwise, if the new slope is greater than the old, we update the slope.

if self.slope(left[(lowerTan.indexL+1) % len(left)],right[lowerTan.indexR]) > lowerTan.slope:

#self.eraseTan(left[lowerTan.indexL],right[lowerTan.indexR])

lowerTan.indexL = (lowerTan.indexL+1) % len(left)

lowerTan.slope = self.slope(left[lowerTan.indexL], right[lowerTan.indexR])

#self.showTan(left[lowerTan.indexL],right[lowerTan.indexR])

moved1 = True

else:

moved1 = False

#We move the right point of lower tan counter clockwise, if the new slope is less than old, update

if self.slope(left[lowerTan.indexL], right[(lowerTan.indexR-1) % len(right)]) < lowerTan.slope:

#self.eraseTan(left[lowerTan.indexL],right[lowerTan.indexR])

lowerTan.indexR = (lowerTan.indexR-1) % len(right)

lowerTan.slope = self.slope(left[lowerTan.indexL], right[lowerTan.indexR])

#self.showTan(left[lowerTan.indexL],right[lowerTan.indexR])

moved2 = True

else:

moved2 = False

#combine hulls with the upperTan and lowerTan

newHull = []

#gets leftmost of left to first point of uppertan

newHull.extend(left[0:upperTan.indexL+1])

#special cases when combining the rest of the hull

if lowerTan.indexR < upperTan.indexR:

newHull.extend(right[upperTan.indexR:])

newHull.extend(right[0:(lowerTan.indexR+1) % len(right)])

else:

newHull.extend(right[upperTan.indexR:lowerTan.indexR + 1])

if lowerTan.indexL != 0:

newHull.extend(left[lowerTan.indexL:])

#find new right most index

newRmi = 0

for i in range(len(newHull)):

if newHull[i].x() > newHull[newRmi].x():

newRmi = i

#erase tans

#self.eraseTan(left[upperTan.indexL], right[upperTan.indexR])

#self.eraseTan(left[lowerTan.indexL], right[lowerTan.indexR])

#erase hulls

#self.eraseHull(left)

#self.eraseHull(right)

return newHull, newRmi

#This is the divide and conquer function

def getHull(self, points):

#Base case: if the hull is less than 4 points we create a base hull

if len(points) < 4:

return self.clockwiseOrder(points)

#We split the hull in 2 and give the left most half of points to getHull

left, rmiL = self.getHull(points[0:math.ceil(len(points)/2)])

#self.showHull(left)

#We take the right half of the hull and give it to getHull

right, rmiR = self.getHull(points[math.ceil(len(points)/2):])

#self.showHull(right)

#We then take both halves of hull and combine them

return self.combine(left, rmiL, right, rmiR)

#End my functions

1. Time and Space Complexity

Time

The overall time complexity of the algorithm is O(n \* log(n)) where n is the number of points.

The divide-and-conquer function “getHull” divides the number of points in two parts, each one half the size of the original array, and the combine method is puts the hulls together in linear time. Using the Master’s Theorem, a = 2, b = 2 and c = 1. Since log2(2) == 1, logb(a) = c, so the time complexity is O(n \* log(n))

The “combine” method is O(n) because worst case, we iterate through the total number of points (number of points in the left hull + number of points in the right hull) to find the right most point of the combined hull.

The “clockwiseOrder” method is done in O(n) time in the worst case were we have 3 points and decide which point is next in the clockwise ordering.

The “slope” function is done in constant time.

Space

The overall space complexity of the algorithm is O(n) where n is the number of points. When we divide, we keep passing the original points to new calls of “getHull”, and split that in half until we have less than 3. This means there are never more than n points existing simultaneously.

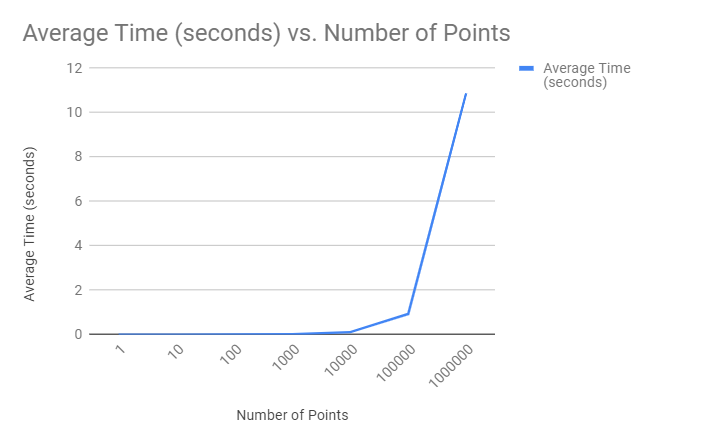
The divide-and-conquer method “getHull” splits the total number of points in half until there are only three, so the most points that exist are n.

The “combine” method only stores 4 extra points worst case which is when the hulls contain all the points of their halves and the tangents also contain two points from each.

The “clockwiseOrder” only uses the number of points it is given so its space complexity is O(n)

The slope function space complexity is O(n) where n is 2.

1. Raw experimental outcome



After taking the average of ten tests for each respective number of points, I found the trend above.

The pattern of the plot most closely follows n \* log(n)

1. Screenshots

