2D Convex Hull Parallel Algorithm

Hazem Nomer

Introduction

This an implementation of the Graham scan divide and conquer 2D convex hull algorithm in python. The algorithm sorts all points based on x-axis. For each point, it is first determined whether traveling from the two points immediately preceding this point constitutes making a left turn or a right turn. If it is not a right turn the second-to-last point is not part of the convex hull and then we delete it. We repeat these steps for both upper and lower parts of convex hull.

Input. A set P of points in the plane.
Output. A list containing the vertices of $\mathcal{CH}(P)$ in clockwise order.

1. Sort the points by x-coordinate, resulting in a sequence $p_1, \ldots, p_n$.
2. Put the points p_1 and p_2 in a list $\mathcal{L}_{\text{upper}}$, with p_1 as the first point.
3. **for** $i \leftarrow 3$ **to** n
4. **do** Append p_i to $\mathcal{L}_{\text{upper}}$.
5. **while** $\mathcal{L}_{\text{upper}}$ contains more than two points **and** the last three points in $\mathcal{L}_{\text{upper}}$ do not make a right turn
6. **do** Delete the middle of the last three points from $\mathcal{L}_{\text{upper}}$.
7. Put the points p_n and p_{n-1} in a list $\mathcal{L}_{\text{lower}}$, with p_n as the first point.
8. **for** $i \leftarrow n-2$ **downto** 1
9. **do** Append p_i to $\mathcal{L}_{\text{lower}}$.
10. **while** $\mathcal{L}_{\text{lower}}$ contains more than 2 points **and** the last three points in $\mathcal{L}_{\text{lower}}$ do not make a right turn
11. **do** Delete the middle of the last three points from $\mathcal{L}_{\text{lower}}$.
12. Remove the first and the last point from $\mathcal{L}_{\text{lower}}$ to avoid duplication of the points where the upper and lower hull meet.
13. Append $\mathcal{L}_{\text{lower}}$ to $\mathcal{L}_{\text{upper}}$, and call the resulting list $\mathcal{L}$.
14. **return** $\mathcal{L}$

2. Python Multiprocessing Module

It is a python package that contains some reasonable support for creating and running multiple processes implemented in Python and for communicating between those processes using Queues and Pipes.
We used this package to create two different processes for the upper and lower convex hulls. Each take the same points. The result is returned in a queue (two different lists). We remove the duplicate points as shown in algorithm from the lower list, and then append both lists

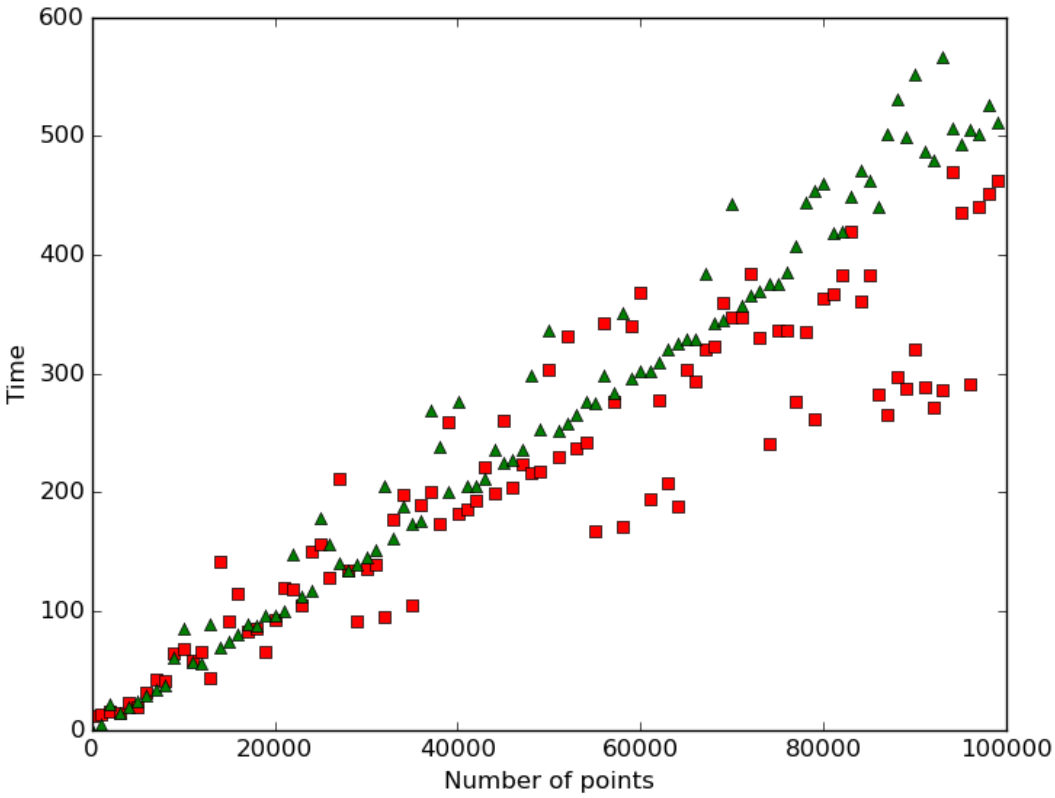
together.
The following is the code of both convex hull sequential algorithm and in parallel. The code generate different number of points and compare between the time taken by both sequential and parallel algorithms. The Pygame module is used to draw points and the convex hull.

```
import random
import pygame
from multiprocessing import Queue , Process
import os
import time
from  matplotlib import pyplot
current_milli_time = lambda: int(round(time.time() * 1000))
def generateRandomPoints(n):
    P = []
    for i in range(n):
        x= random.randint(0,1024)
        y= random.randint(0,720)
        P.append((x,y))
    points = map(None,P)
    points.sort()
    return points
def Det(p, q, r):
    sum1 = q[0]*r[1] + p[0]*q[1] + r[0]*p[1]
    sum2 = q[0]*p[1] + r[0]*q[1] + p[0]*r[1]
    return sum1 - sum2
def isRightTurn((p, q, r)):
    #assert p != q and q != r and p != r
    if Det(p, q, r) < 0:
        return 1
    else:
        return 0
def ConvexHullUpper(P,ans):
    #info("upper convexhull")
    upper=[]
    upper.append(P[0])
    upper.append(P[1])
    for i in range(3,len(P)):
        upper.append(P[i])
        l = len(upper)
        while l > 2 and isRightTurn((upper[l-1], upper[l-2], upper[l-
3]))==0:
            del upper[l-2]
            l = len(upper)
    ans.put(upper)
def ConvexHullLower(P,ans):
    #info("lower convexhull")
    lower =[]
    lower.append(P[len(P)-1])
    lower.append(P[len(P)-2])
    for j in range(len(P)-2,0,-1):
        lower.append(P[j])
        l = len(lower)
        while(l >2 and isRightTurn((lower[l-1], lower[l-2], lower[l-
3]))==0):
            del lower[l-2]
            l = len(lower)
    ans.put(lower)
```

```
def info(title):
    print title
    if hasattr(os, 'getppid'):  # only available on Unix
        print 'parent process:', os.getppid()
    print 'process id:', os.getpid()
def ConvexHullSeq(P):
    t1 = current_milli_time()
    upper=[]
    upper.append(P[0])
    upper.append(P[1])
    for i in range(3,len(P)):
        upper.append(P[i])
        l = len(upper)
        while l > 2 and isRightTurn((upper[l-1], upper[l-2], upper[l-
3]))==0:
            del upper[l-2]
            l = len(upper)
    lower =[]
    lower.append(P[len(P)-1])
    lower.append(P[len(P)-2])
    for j in range(len(P)-2,0,-1):
        lower.append(P[j])
        l = len(lower)
        while(l >2 and isRightTurn((lower[l-1], lower[l-2], lower[l-
3]))==0):
            del lower[l-2]
            l = len(lower)
    del lower[0]
    del lower[-1]
    t2=current_milli_time()
    return tuple(upper + lower) , (t2-t1)
def ConvexHullParallel(P):
    t1 = current_milli_time()
    ans =Queue()
    upperp = Process(target=ConvexHullUpper, args=(P,ans))
    lowerp = Process(target=ConvexHullLower , args=(P,ans))
    upperp.start()
    lowerp.start()
    upperp.join()
    lowerp.join()
    upper = ans.get()
    lower = ans.get()
    del lower[0]
    del lower[-1]
    t2=current_milli_time()
    return tuple(upper + lower) ,t2-t1
#exp
time_parallel =[]
time_seq=[]
pointsn=[]
for i in range(10,100000,1000):
    points = generateRandomPoints(i)
    [sp,tp]=  ConvexHullParallel(points)
    [sq,tq]= ConvexHullSeq(points)
    time_parallel.append(tp)
    time_seq.append(tq)
    pointsn.append(i)
```

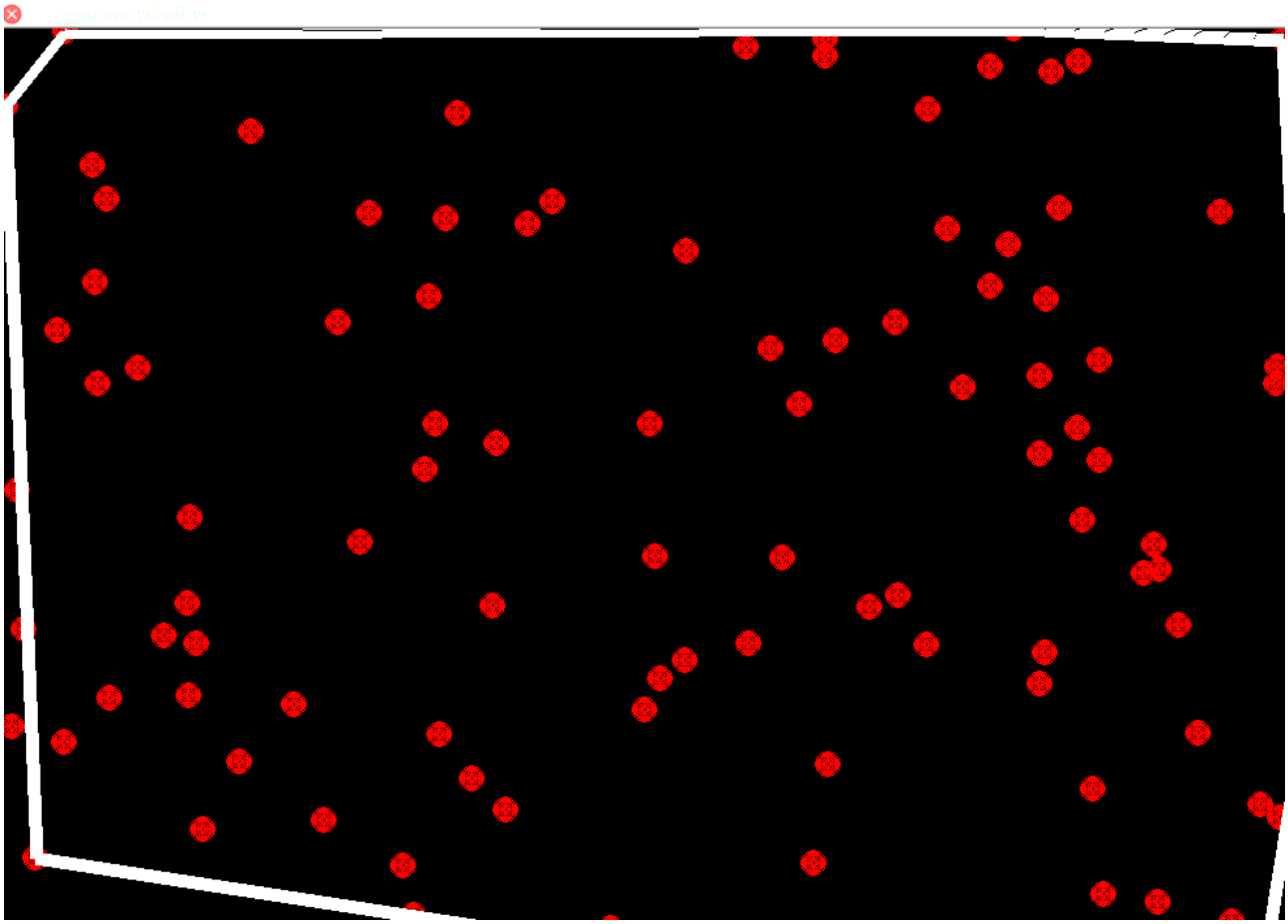
```
print len(pointsn) , len(time_parallel)
pyplot.xlabel('Number of points')
pyplot.ylabel('Time')
pyplot.plot(pointsn,time_parallel, "rs" , pointsn , time_seq,"g^")
pyplot.show()
points = generateRandomPoints(100)
s,t= ConvexHullParallel(points)
pygame.init()
screen = pygame.display.set_mode((1024,720))
red = (255,0,0)
white = (255,255,255)
for p in points:
    pygame.draw.circle(screen, red , p , 10, 10)
    pygame.display.update()
s = list(s)
pygame.draw.lines(screen,white , True, s, 10)
pygame.display.update()
running = True
while running:
  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      running = False
```

3. Results



The graph represents the relation between number of points and time. The time taken by sequential algorithm is represented as green triangles and the time of parallel algorithm is represented as red squares. You can see that time of sequential algorithm increases when the number of points increases compared to the time of parallel algorithm. For lower number of points (< 30000), the sequential algorithm takes less time than that of parallel one. That is

due to the overhead of creation of processes and communication between them. The overhead is neglected when the input increases.



Convex hull of 100 points using parallel algorithm