ALGORITMO DE ENVOLVENTE CONVEXA PARALELIZADO EN CUDA
ALGORITHM OF ENVELOPE CONVEXA PARALLELIZED IN CUDA

Juan Pablo Florez Caicedo, Christhian Julián Gómez Castaño
Pregrado en ingeniera de sistemas y computación, Universidad Tecnológica de Pereira, Pereira, Colombia
Correo-e: junpaflorez@utp.edu.co, chjugomez@utp.edu.co

Resumen— Existen diferentes metodologías para hallar el envolvente convexo de un conjunto finito de puntos en un plano que forman un polígono, en este documento evaluaremos la implementación de uno de estos métodos en CUDA al ser una herramienta que nos brinda paralelismo y así lograr un mejor rendimiento. Primero explicaremos 3 algoritmos para obtener la envolvente convexa de un conjunto finito de puntos (Quick hull, Gift wrapping y Graham Scan) y luego daremos reporte completo del método que se utilizó. Entre las distintas aplicaciones que se realizan con el convex hull cabe resaltar las siguientes: planificación del espacio comercial en una ciudad, concentración de puntos de interés en una región, entre otras.

Palabras clave— Envolvente Convexo, Gift Wrapping, Graham Scan, Quick Hull, CUDA, Merge Sort.

Abstract— There are different methodologies to find the convex envelope of a finite set of points in a plane that form a polygon, in this document we will evaluate the implementation of one of these methods in CUDA as it is a tool that gives us parallelism and thus achieve better performance. First we will explain 3 algorithms to obtain the convex envelope of a finite set of points (Quick hull, Gift wrapping and Graham Scan) and then we will give full report of the method that was used. Among the different applications that are made with the convex hull, it is worth mentioning the following: planning of the commercial space in a city, concentration of points of interest in a region, among others.

Key Word — Convex Envelope, Gift Wrapping, Graham Scan, Quick Hull, CUDA, Merge Sort.

I. INTRODUCCIÓN

La envolvente convexa es un pequeña región que incluye un grupo de puntos específicos, El borde o frontera de los mismos es un polígono cerrado convexo; La envolvente convexa puede ser vista como una banda elástica alrededor de los puntos de la envolvente, y los otros puntos como si estuvieran dentro de esta banda elástica; La envolvente convexa juega un papel central en el campo de la geometría computacional. Este concepto geométrico ofrece aplicaciones prácticas en muchas aéreas ya que puede ser usada computacionalmente para fines tales como procesamiento de imágenes, reconocimiento de patrones, sistemas de información geográfica, automatización de diseño e investigación de operaciones entre otros. [1][3].

Gracias al paralelismo se pueden obtener resultados muy interesantes a diferentes tipos de algoritmos, tales como el procesamiento de imágenes, las redes neuronales, entre otras. De igual manera se pretende mejorar el tiempo de procesamiento de la geometría computacional, mediante implementaciones en CUDA. Este tipo de implementación puede tener diferentes varianzas según lo que se pretende paralelizar, por ejemplo el manejo del conjunto de puntos, es una forma práctica para implementar la computación paralela, esta estrategia ayudara considerablemente el tiempo de procesamiento de los algoritmos, puesto que resuelve el principal problema de la geometría computacional, ya que los tiempos de comparación entre puntos es lo que más consume tiempo de procesamiento.

II. CONTENIDO

Vamos a describir cada uno de los Algoritmos

Algoritmo Quick-Hull: Es un algoritmo rápido que tarda en tiempo de cómputo O (n log n), para encontrar la envolvente y en el peor de los casos O (n2).

Una idea muy efectiva para acelerar el proceso de cálculo de la envolvente convexa, consiste en desechar la mayor cantidad de puntos interiores, pues ´estos no califican para la envolvente.

Como funciona:
Iniciamos el proceso dividiendo al conjunto S en dos partes casi iguales, mediante una recta horizontal L. Denotamos las partes por S1 y S2. Tanto S1 como S2 contienen un gran triangulo cuyos vértices forman parte de la envolvente convexa de S1 (resp. de S2). Entonces este triángulo será una primera aproximación de la envolvente convexa de S1 (resp. de S2). Ver la figura.

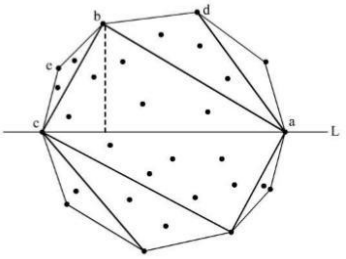


Figura 1. Quick Hull

El punto c (respec. a) es aquel cuya coordenada X es mínima (respec. máximo). En la figura la recta L divide a S en dos partes, una arriba y la otra hacia abajo; S1 y S2. El punto b es el punto de S1 más alejado de la recta L en

la dirección perpendicular a L. El triángulo abc es una primera aproximación de la envolvente convexa de S1. Los puntos interiores los desechamos.

El punto d es el más alejado de la recta ***ba*** dentro de **S1**

El punto e es el más alejado de la recta ***bc*** dentro de **S1**

Algoritmo Gift Wrapping

También es conocido como Marcha de Jarvis,; tiene costo de O (nh), donde n es el número de puntos y h es el número de puntos en la envolvente convexo. Su rendimiento real en comparación con otros algoritmos de esta clase es favorable cuando n es pequeño o se espera que h sea muy pequeño con respecto a n. En general, puede ser superado por otros algoritmos.

Como Funciona:

Supongamos que conocemos un lado e de la envolvente, el cual yace en posición horizontal en la parte de abajo y cuyo extremo derecho es igual al punto x. Sabemos que x debe estar conectado con el siguiente punto de la envolvente, cuando este polígono se recorre en sentido contrario a las agujas del reloj. Llamaremos a este punto hipotético y. La pregunta es: ¿Cómo hacemos para calcular *y*?

La idea consiste en trazar una línea L en el plano que tiene un extremo pivote en x, y la cual hacemos girar en sentido anti horario, observando el ángulo que forma con el lado e. Entonces el primer punto de S que toca esta línea es y. Este paso se lleva a cabo considerando todas las rectas del tipo ***Lxs***
con s ∈ S. Podemos entonces comparar las pendientes de todas ellas y tomamos el punto y en S tal que la pendiente de
Lxy sea mínima. (ver la figura)

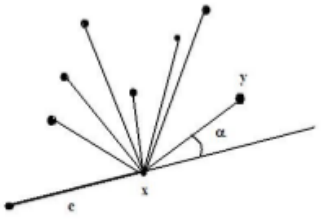


Figura 2. Gift Wrapping

En realidad, en la etapa de implementación del algoritmo, no es necesario calcular en ningún momento las pendientes de estas rectas. Calcular ´ángulos requiere usar aritmética de números reales, funciones trigonométricas y por lo tanto caer en errores de redondeo y otro tipo de problemas que debilitan a los algoritmos. Una forma bastante delicada de hacer esto, es comparar las pendientes usando nuestra

Luego la poligonal ***adbec*** es otra aproximación a la envolvente convexa de S1. Como se ve, después de aplicar recursivamente este algoritmo, un número finito de pasos, llegamos a la envolvente convexa de S1 y S2. El paso consiste en concatenar o unir las dos envolventes para obtener la envolvente de S. [3]
fórmula de área signada para un triángulo. Es fácil ver entonces que la recta ***xa*** tiene menor pendiente que la recta ***xb*** sí y sólo si el ´área del triángulo Δ***(xba***) es positiva.

A medida que vamos avanzando, vamos caminando sobre todos los puntos de la envolvente y por esto se llama también a este algoritmo Marcha de Jarvis. Al llegar a la parte más alta del recorrido, los ´ángulos empiezan a ser negativos y los medimos de arriba hacia abajo. Después que la línea de barrido ha hecho un giro completo de 3600 volvemos a la posición inicial y se ha completado el algoritmo. Finalmente, para hallar el primer punto ***x*** en este proceso, ordenamos todos los puntos de S de acuerdo a su coordenada **X**, y tomamos el mínimo. Esto tiene un costo de O(nlogn)

Algoritmo Graham Scan

Es uno de los algoritmos utilizados para calcular la envolvente convexa. Requiere en tiempo de computo O (n) y en el peor caso de O (nlgn) (es decir, nlog2 n) debido a un paso de clasificación inicial que ordena los puntos en coordenadas cartesianas]. Graham Scan verifica regularmente la lista ordenada de puntos y elimina la envolvente convexa de puntos extremos. [1]

Como funciona:

Tomamos un punto sobre la envolvente y ordenamos en forma radial todos los puntos de S. Al hacer el recorrido en este orden vamos conectando con lados los puntos adyacentes y de esta manera vamos construyendo un polígono. Cuando un punto sea del tipo reflex, entonces lo eliminamos.

Para iniciar el algoritmo elegimos el primer punto de S, denotado por p0, como el punto de mas baja altura. En caso de empate, se toma el del extremo derecho. Luego ordenamos todos los puntos de S en el sentido contrario a las agujas del reloj de acuerdo al ángulo que forma en el rayo que parte de p0 y termina en el punto. Ver figura: Entonces p1 es un punto de la envolvente, al igual que p2, pues p0p1p2 es un giro a la izquierda.

Continuando de esta manera, podemos decir que p3 es también de la envolvente, pues p1p2p3 es un giro a la izquierda. Sin embargo, en el siguiente paso vemos que p2p3p4 es un giro a la derecha. Por lo tanto eliminamos el punto p3 y nos quedamos con los puntos p0,p1,p2,p4. Para el paso siguiente vemos que p2p4p5 es un giro a la derecha y por lo tanto eliminamos al punto p4 y nos quedamos con p5. Continuando de esta manera, después de dar un

recorrido completo anti horario, llegamos al punto de inicio p0.

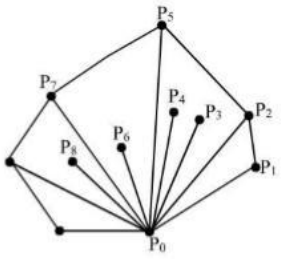


Figura 3. Graham Scan

La pila (stack) es una estructura de datos muy usada que permite guardar objetos como número, puntos, figuras, etc. Comenzamos a almacenar los objetos en el fondo (Bottom) de la pila, colocándolos unos encima de otros. El último objeto almacenado se encuentra en el tope (Top). Con una pila S se manejan dos operaciones básicas

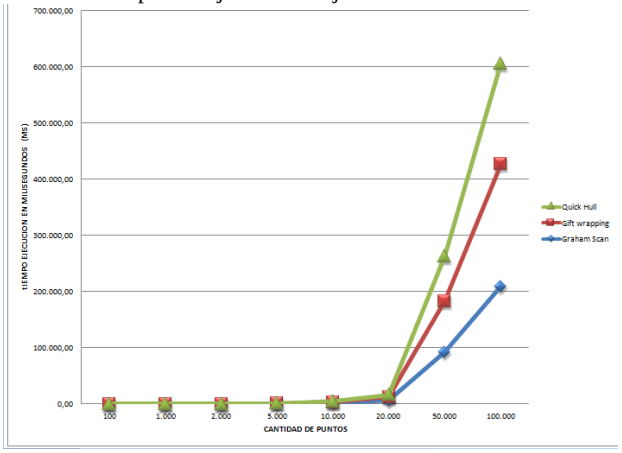
1. PUSH (p,S), la cual inserta un objeto p en el tope de la pila
2. POP (S), mediante la cual se elimina el objeto que se encuentra en el tope de la pila.

Implementación Prueba de los algoritmos, los tiempos son en ms estos son sin computación paralela.

TABLA I Comparativo en Java en ms

PUNTOS	Quick Hull	Gift Wrapping	Graham Scan
100	1,1	18,8	2,0
1000	11,8	27,5	17,7
2000	36,6	116,1	48,2
5000	351,7	394	377
10000	1.590,7	1.484,5	1.541,9
20000	3.772,1	5.689	6.736,3
50000	79.446,7	90.943,1	92.470,8
100000	179.562,4	217.274,3	209.456,9

Tabla 1. Tiempos de ejecución en java.



Paralelismo en CUDA:

CUDA es una arquitectura de cálculo paralelo de NVIDIA que aprovecha la gran potencia de la GPU (unidad de procesamiento gráfico) para proporcionar un incremento extraordinario del rendimiento del sistema.

La plataforma de cálculo paralelo CUDA proporciona unas cuantas extensiones de C y C++ que permiten implementar el paralelismo en el procesamiento de tareas y datos. [5]

El desarrollo en CUDA, permite llegar más allá de un simple procesamiento sobre la CPU, este tipo de implementaciones Funciona en todas las GPU nVidia de la serie G8X en adelante, incluyendo GeForce, Quadro, ION y la línea Tesla.

Implementando CUDA en la Geometría Computacional:

Para la implementación de algoritmos como los mencionados, evaluamos las diferentes formas en las que se podrían paralelizar, tales como el manejo del conjunto de puntos, el manejo de los ciclos o pilas concretos de cada algoritmo. Para tomar decisión sobre en donde deberíamos fijar la computación paralela, atacamos el problema general entre los 3 algoritmos anteriormente mencionados, que es el manejo del conjunto de puntos, permitiendo reducir en gran medida el costo computacional, sin importar la complejidad computacional de cada uno de los algoritmos.

Una vez decidido, la siguiente tarea es implementarlo en uno de los 3 algoritmos, La envolvente convexa implementada con Graham Scan obtiene excelentes resultados a comparación de las otras 2. Por lo que esta fue la elegida. En el desarrollo de la misma encontramos que al usar el método de ordenamiento Merge Sort, la implementación de la computación paralela no obtiene los resultados esperados, encontramos que en el diseño que realizamos, primero estaba la existencia del problema de la divergencia en el paralelismo, por lo que no se aprovecharon correctamente los hilos, además de que en la implementación el último par de hilos que se encargaba de re ordenar los puntos, tenía que revisar todo el conjunto de puntos y esto implica que no se mejore el tiempo de procesamiento, como recomendaciones para resolver el problema del ordenamiento de los puntos, es importante tener técnicas de ordenamiento más prácticas y rápidas, como lo son ordenamiento binario y las Quick Sort.

III. CONCLUSIONES

- Los algoritmos que se ejecutaron en java se corrieron 50 veces cada uno de ellos con (100, 1.000, 2.000, 5.000, 10.000 y 20.000 puntos), los 50.000 y 100.000 puntos fueron ejecutados 10 veces por cada uno de los algoritmos debido a que

la ejecución de estos tiene un alto costo computación.

- El uso de Merge Sort no fue la mejor para ordenar los puntos, por lo que es bueno observar diferentes técnicas de ordenamiento y considerar si el uso de los hilos realmente ayuda a mejorar los tiempos de procesamiento en la técnica que se vaya a utilizar.
- Consideramos que el paralelismo en MPI, podría ser una buena solución para este problema presente en la geometría computacional.

RECOMENDACIONES

Los algoritmos se ejecutaron sin la parte gráfica y sin el cálculo del área para que esta no afectara el tiempo de ejecución.

Es recomendable, mantener un mínimo de procesos en ejecución en el procesador porque esto afecta los tiempos del algoritmo.

Las técnicas de ordenamiento son las que más tiempo de análisis consumen, es importante observar los pro y contras de cada una de las técnicas seleccionadas, para valorar correctamente cual de todas es la más acorde a cada algoritmo.

REFERENCIAS

[1] **Muhammad Sharif, Safdar Khan, Sadaf Jameel Khan, Mudassar Raza, An Algorithm to Find Convex Hull Based on Binary Tree, · December 2009.**

[**2] Maher M. Atwah , Johnnie W. Baker and Selim Akl, An Associative Implementation Of Graham's Convex Hull Algorithm, January 1995**.

[3] Rivero F, Geometría Computacional, Venezuela

[4]Borgwardt K. H, Average Complexity of a Gift-Wrapping Algorithm for Determining the Convex Hull of Randomly Given Points, 1997.

[5] Nvidea

[6] David B.Kirk, Wen-Mei W Hwu, Programming Massively Parallel Procesors, second Edition. 255 Wyman Street, Whaltam, Ma, 2013.