DBSCAN modificado con Octrees para agrupar nubes de puntos en tiempo real

Octavio Navarro-Hinojosa, Moisés Alencastre-Miranda

Tecnologico de Monterrey, Campus Santa Fe, Ciudad de México, México

A00967953@itesm.mx, malencastre@itesm.mx

Resumen. Con el auge de sensores de profundidad comerciales, como el Kinect, se crean nuevas oportunidades para desarrollar aplicaciones y sistemas interactivos que utilicen el cuerpo humano. Sin embargo, esos sensores generan una gran cantidad de datos en 3D (nubes de puntos) que tienen que ser procesados, buscando obtener obtener información relevante para aplicaciones específicas. Los algoritmos de agrupamiento, usualmente usados para minería de datos, son útiles para descubrir esa información. Uno de los más conocidos es DBSCAN, que permite generar un número desconocido de grupos en un conjunto de datos, al mismo tiempo que filtra ruido. Sin embargo, también puede ser lento debido al tipo de dato que se use, así como a la búsqueda de datos con características similares. En este trabajo, se propone el uso de DBSCAN para procesar nubes de puntos. Adicionalmente, se propone una modificación del algoritmo, utilizando octrees para acelerar la búsqueda de vecinos que realiza, así como un esquema de particionamiento para que no se tengan que hacer búsquedas de vecinos de todos los puntos de una nube. Con el método propuesto, se logró acelerar considerablemente el procesamiento de nubes de puntos en comparación con el algoritmo original, logrando procesamiento en tiempo real, obteniendo los mismos resultados.

Palabras clave: DBSCAN, particionamiento de datos, estructuras de datos espaciales, datos de profundidad, aplicaciones interactivas.

Modification of the DBSCAN Algorithm with Octrees in order to Cluster Point Clouds in Real Time

Abstract. With the advent of commercial depth sensors, such as the Kinect, new opportunities to develop applications and interactive systems that use the human body are created. However, those sensors capture a large amount of 3D data, known as point clouds, that have to be processed, trying to obtain as much relevant information as possible. Clustering algorithms, normally used in data mining, are useful to

discover that information. One of the most known is DBSCAN, which allows the creation of an unknown number of clusters within a data set, while filtering out noise. Unfortunately, the algorithm can also be slow due to the data type of the data set, as well as because of the search of data with similar characteristics. In this work, the use of the DBSCAN algorithm to process point clouds is proposed. Additionally, a modification of the algorithm is presented: using octrees to decrease the neighbor searching times, and using a data partitioning scheme in order to reduce the number of elements of each search. With the proposed method, the point cloud processing time was considerably accelerated when compared to the original algorithm, achieving real time processing, while obtaining the same results.

Keywords: DBSCAN, data partitioning, spatial data structures, depth data, interactive applications.

1. Introducción

En los últimos años, la introducción de sensores de profundidad ha creado nuevas oportunidades para generar experiencias y aplicaciones interactivas en las que se utiliza el cuerpo humano. El Kinect de Microsoft [14] fue uno de los primeros sensores de profundidad comerciales de acceso general disponibles en el mercado. Aunque su principal uso fue para jugar videojuegos interactivos utilizando el cuerpo [16], éste le permitió a desarrolladores e investigadores explorar el uso de reconocimiento de movimiento en áreas como salud, educación, entretenimiento, arte, robótica, reconocimiento de gestos, reconstrucción 3D, entre otras [12].

Usando un sensor infrarrojo, el Kinect genera un flujo de datos de profundidad que sirve para construir nubes de puntos en tres dimensiones que representan un lugar físico. Dependiendo de la aplicación final, el volumen de datos en una nube que se obtiene de un solo punto en el tiempo puede ser considerablemente alto, hasta un total de 307200 puntos por cuadro [14]. Al desarrollar aplicaciones interactivas, es necesario encontrar métodos que permitan procesar ese volumen de datos en tiempo real, es decir, procesar al menos 30 cuadros por segundo, y que permitan obtener información relevante. Una de las técnicas que logran esto, es el análisis de grupos (clustering analysis).

El análisis de grupos es una técnica en minería de datos que divide un conjunto de datos en grupos, cada uno con características similares, y ayuda a obtener información adicional de dichos grupos. Un algoritmo de agrupamiento descubre esos grupos en los datos al maximizar las similitudes entre un grupo de objetos, y minimizarlo entre los grupos que se generen. Algunos de los campos de aplicación para el análisis de grupos son análisis estadístico, reconocimiento de patrones, procesamiento de imágenes, entre otros [18].

Los algoritmos de agrupamiento se pueden dividir en cuatro tipos: particionamiento, jerárquicos, basados en densidad, y basados en mallas [8]. DBSCAN (Density Based Spatial Clustering of Applications with Noise) es un algoritmo

de agrupamiento basado en densidad. La idea principal detrás del algoritmo es que por cada dato en un grupo, el vecindario dentro de un radio determinado (*eps*) tiene que contener al menos un número mínimo de puntos (*minpts*), es decir, la densidad de un vecindario tiene que exceder un umbral específico [5].

Uno de los problemas de DBSCAN es que debido a su complejidad computacional de $O(n^2)$ [6] no es eficiente para procesar un gran número de datos, como las nubes de puntos que arroja el Kinect, en tiempo real. Otro punto importante es que la mayoría del tiempo que utiliza el algoritmo para el proceso de agrupamiento es usado en las operaciones de búsqueda de vecinos [18].

En el presente trabajo, se tiene como objetivo realizar modificaciones al algoritmo DBSCAN para lograr que procese nubes de puntos, obtenidas de un sensor de profundidad, en tiempo real, y poder utilizarlo en aplicaciones interactivas como videojuegos. Se propone el uso de Octrees (estructuras de datos en las que cada nodo tiene exactamente ocho hijos, que sirven para subdividir un espacio en tres dimensiones) [13] para acelerar la búsqueda de vecinos del algoritmo. De igual forma, se propone un nuevo esquema de particionamiento con el que se reduce considerablemente el espacio de búsqueda del algoritmo, al mismo tiempo que se obtienen los mismos grupos que con el algoritmo original.

El resto de este trabajo se organiza como sigue. La siguiente sección muestra trabajo relacionado. Después se presenta un resumen del algoritmo original, y se analizan sus limitaciones al trabajar con un gran volumen de datos. De igual forma, se presentará la idea básica de los Octrees, y el por qué de su uso para mejorar el desempeño de DBSCAN. Posteriormente, se introduce el algoritmo modificado indicando cómo se integra el uso de octrees, así como el esquema de particionamiento. Después, se mostrarán resultados experimentales para demostrar la efectividad del algoritmo propuesto al aplicarlo para agrupar nubes de profundidad. Finalmente se concluye y se comenta el trabajo a futuro.

2. Trabajo Relacionado

El algoritmo DBSCAN, propuesto por Ester et. al. [5], está basado en la detección de grupos al comparar las densidades entre los puntos de un conjunto determinado. Aunque es conceptualmente sencillo, ha probado ser efectivo para identificar grupos en grandes bases de datos. En los últimos años, muchos trabajos basados en DBSCAN han sido propuestos y aplicados en diferentes áreas. De igual forma, han habido muchos trabajos que buscan mejorar el desempeño del algoritmo.

Zhou et. al. [18] propone varios enfoques para mejorar el desempeño de DBSCAN y poder aplicarlo a bases de datos grandes. Entre ellos, una modificación basada en muestras, otra basada en particionamiento, y un algoritmo paralelo. Arlia et. al. [1] define un algoritmo en paralelo basado en un esquema de esclavo y maestro, donde el maestro se encarga de hacer la asignación de grupos, mientras el esclavo realiza consultas de vecindad sobre un árbol R*. Götz et.al. [8] presentan una versión en paralelo y escalable de DBSCAN. Su trabajo se basa en un esquema de dividir y conquistar, usando técnicas de algoritmos

de agrupamiento basados en celdas. Específicamente, utilizan una hiper-malla para minimizar la búsqueda de vecinos, y para particionar el agrupamiento en subtareas. Finalmente, proponen un esquema de unión para combinar los grupos que se obtuvieron en las subtareas. Thapa et. al. [17] proponen un enfoque con tarjetas de video programables (GPU, por sus siglas en inglés), para procesar DBSCAN en paralelo. Su trabajo permite una mejor escalabilidad de memoria, lo que les permite usar el algoritmo con bases de datos muy grandes. De igual forma, se mejora considerablemente el tiempo de ejecución del algoritmo.

Zhou et. al. [19] introduce un nuevo esquema para agrupar basado en árboles de búsqueda digital, entre ellos quadtrees y octrees. Su enfoque ayudó a reducir los espacios de búsqueda que los algoritmos de agrupamiento usualmente tienen que recorrer para agrupar los datos. Principalmente ayudan a evitar dimensiones elevadas de búsqueda, y eliminar unidades vacías en las que se hubiera tenido que hacer búsquedas. Joshi et. al. [10] proponen un algoritmo basado en DBSCAN, para agrupar polígonos en un espacio. Para agrupar polígonos, se incorporan sus propiedades topológicas y espaciales en el proceso de agrupamiento al usar una función de distancia adecuada para el espacio de los polígonos. Su objetivo es crear grupos compactos de polígonos.

En el trabajo de Borah et. al. [3], se presenta una mejora para DBSCAN basada en un muestreo de los datos de una base de datos. Su método reduce el uso de memoria y el tiempo de ejecución, pero mantiene la calidad de los grupos generados. En [11], Liu propone una modificación a DBSCAN en la que se busca mejorar el tiempo de ejecución de las búsquedas por región del algoritmo, al seleccionar ciertos puntos ordenados sin marcar que están fuera del vecindario de los puntos núcleo, y con ellos expandir los grupos. Este enfoque mejora tanto los tiempos de ejecución como la calidad de los grupos encontrados.

Bianchi y Martinelly [2] utilizan DBSCAN para estimar, en tiempo real, la forma y posición de objetos en una nube de puntos de profundidad. Ya que DBSCAN da buenos resultados, pero es caro computacionalmente y no permite tiempo real, se utilizó un subconjunto de los datos, seleccionado con filtros de imágenes basados en convolución, para acelerar el agrupamiento de los puntos. En el presentado por Doan et. al. [4] se busca hacer una segmentación de objetos encontrados en nubes de puntos de profundidad utilizando DBSCAN para extraer los grupos que representen un objeto. Su objetivo es desarrollar aplicaciones de realidad aumentada, usando sensores de profundidad, para ofrecer una mayor inmersión a los usuarios al modificar los objetos de la escena. Ghosh y Lohani [7] describen un trabajo en el que se analizan nubes de puntos de profundidad obtenidas con tecnología LIDAR aérea, para generar conjuntos de datos densos y precisos de terreno. Se utiliza el método de DBSCAN para separar los datos en grupos distintos de terreno, basándose en reglas topográficas, que serán procesados posteriormente.

Aunque la mayoría de estos trabajos logran reducir considerablemente el tiempo de procesamiento de DBSCAN, solo Götz et.al. [8] logran mejoras en tiempo suficientes para procesar nubes de puntos en tiempo real. Sin embargo, su solución requiere de una gran cantidad de recursos computacionales: sus experi-

mentos y tiempos reportados fueron realizados con 768 núcleos de procesamiento, de un sistema con 2472 núcleos.

El presente trabajo busca mejorar el desempeño de DBSCAN al utilizar octrees para reducir el tiempo de las búsquedas de vecinos, así como mediante el uso de un esquema de particionamiento, para reducir la cantidad de puntos que van a ser procesados. El objetivo es reducir el tiempo de procesamiento de DBSCAN lo suficiente como para poder utilizarlo en aplicaciones en tiempo real.

3. El algoritmo DBSCAN

DBSCAN es un algoritmo de agrupamiento basado en densidad publicado por Ester et al. [5]. Está diseñado para descubrir grupos de forma arbitraria mientras que es capaz de manejar con eficacia el ruido y los valores atípicos. La idea principal es la de encontrar áreas densas y expandirlas recursivamente para encontrar grupos. Un área densa está formada por un punto que tiene al menos *minPts* puntos vecinos, dentro de un radio de búsqueda ε. Esta área densa también es llamada el *núcleo* de un grupo. Se aplica el concepto previo a cada uno de los vecinos, y el grupo se expande. Todos los puntos que no formen un núcleo y que no son "absorbidos" por este proceso de expansión se consideran como ruido. Las siguientes definiciones describen el algoritmo con respecto a sus parámetros ε y *minPts* [8]. En la Figura 1 se pueden ver ilustradas estas definiciones.

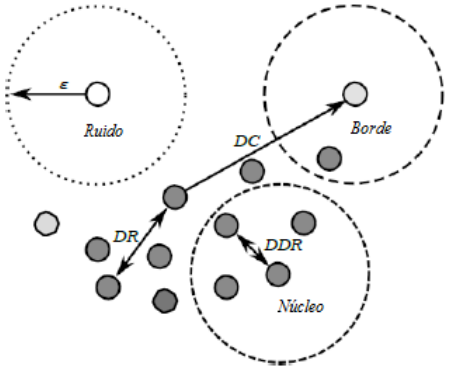

Fig. 1: Agrupamiento con DBSCAN con $minPts = 4$. Reproducido de [8]

Definición 1. Vecindario Epsilon N_ε: El vecindario epsilon N_ε de p denota todos los puntos q de un conjunto de datos X, que tienen una distancia $dist(p, q)$ que es mejor o igual a ε. Es decir $N_\varepsilon(p) = \{q | dist(p, q) \leq \varepsilon\}$. En la práctica, se usa normalmente la distancia euclidiana para evaluar $dist$.

Definición 2. El punto núcleo: p es considerado un punto núcleo si el vecindario epsilon de p contiene al menos $minPts$ puntos incluyéndose a sí mismo: $Core(p) = |N_\varepsilon(p)| \geq minPts$.

Definición 3. Directamente alcanzable por densidad (Directly Density Reachable, DDR): Un punto q es es directamente alcanzable por densidad desde un punto p, si p está dentro del vecindario epsilon de q y p es un punto núcleo: $DDR(p, q) = q \in N_\varepsilon(p) \wedge Core(p)$.

Definición 4. Alcanzable por densidad (Density Reachable, DR): Un par de puntos $p_0 = p$ y $p_n = q$ se llaman alcanzables por densidad si existe una cadena de puntos directamente alcanzables por densidad -$\{p_i|0 \leq i \wedge i < n \wedge DDR(p_i, p_{i+1})\}$-que los une uno al otro.

Definición 5. Punto borde: Los puntos borde son puntos dentro del grupo que usualmente se encuentran en los extremos. Estos no cumplen el criterio de punto núcleo pero aún así se incluyen en el grupo debido a que son directamente alcanzables por densidad. Formalmente, esto se expresa como: $Border(p) = |N_\varepsilon(p)| < minPts \wedge \exists q : DDR(q, p)$.

Definición 6. Conectado por densidad: Dos puntos p y q se dicen conectados por densidad, si hay un tercer punto r, tal que r pueda alcanzar por densidad a p y q: $DC(p, q) = \exists r \in X : DR(r, p) \wedge DR(r, q)$.

Definición 7. Grupo: Un grupo es un subconjunto de un conjunto de datos, donde cada uno de los puntos es alcanzable por densidad a todos los otros del grupo, y que contiene al menos un punto núcleo. Formalmente esto se define como: $\emptyset \subset C \subseteq X$ con $\forall p, q \in C : DC(p, q)$ y $\exists p \in C : Core(p)$.

Definición 8. Ruido: Los puntos que se consideran como ruido, son puntos que no pertenecen a ningún vecindario epsilon, tal que $Noise(p) = \neg\exists q : DDR(q, p)$.

Para encontrar un grupo, DBSCAN empieza con un punto arbitrario p en X y recupera todos los puntos que sean alcanzables por densidad de p, con respecto a ε y *minPts*. Si p es un punto núcleo, se crea un grupo. Si p es un punto borde, no se alcanzaron puntos por densidad desde p, y p se asigna temporalmente como ruido. Posteriormente, el algoritmo procesa el siguiente punto en el conjunto de datos, y el grupo se va expandiendo al ir recuperando puntos alcanzables por densidad al realizar consultas sucesivas de región.

Considerando que DBSCAN no realiza ningún tipo de pre-agrupamiento, y que opera directamente sobre el conjunto de datos, al trabajar sobre conjuntos de datos grandes, el costo de tiempo para hacer las consultas por región e ir haciendo al agrupamiento de todos los puntos es bastante alto. Esto se puede confirmar con el trabajo de Gunawan [9], donde se expone que el algoritmo tiene una complejidad de $O(n^2)$, principalmente debido a la búsqueda de vecinos. Si se usa una estructura de indexado, como son los árboles R*, el desempeño se mejora considerablemente, ejecutándose en $O(n \log n)$.

4. Octrees

Un octree [13] es una estructura de datos en la que cada nodo tiene exactamente ocho hijos. Se usan principalmente para particionar un espacio en tres

dimensiones al subdividirlo recursivamente en ocho octantes, hasta que se cumpla una condición. Para el caso de un conjunto de datos en tres dimensiones, una condición podría ser que se siga subdividiendo hasta que solamente exista un punto por cada nodo; otra condición sería que se siga subdividiendo hasta que los nodos tengan una longitud de lado determinada. Estos son los análogos en tres dimensiones de los quadtrees. Un ejemplo de la construcción de un octree se puede ver en la Figura 2.

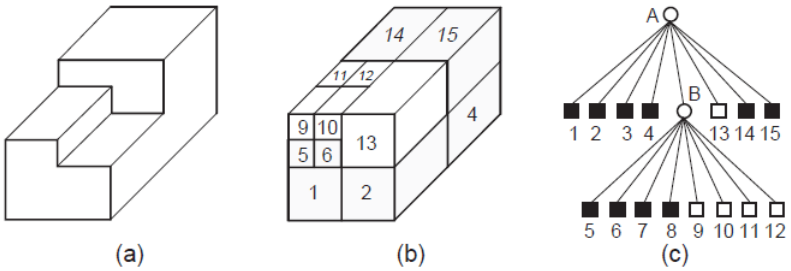

Fig. 2: (a) Ejemplo de un objeto en tres dimensiones; (b) su descomposición en un octree; (c) su representación en forma de árbol. Reproducido de [15]

Un octree es particularmente útil para realizar operaciones sobre conjuntos de datos grandes; una de las principales es la búsqueda de datos basándose en la localización. Esas búsquedas son sencillas de resolver con un octree ya que implican descender por el árbol hasta que se encuentre el objeto que se deseé. Si se desea un vecino de un punto, la búsqueda se continúa en el vecindario del nodo que contiene el objeto [15]. Esta última característica es de principal importancia cuando se considera que el conjunto de datos son nubes de puntos obtenidas de un sensor de profundidad, ya que se están procesando datos en tres dimensiones que representan una posición en el espacio. De igual forma, esta propiedad permite resolver el problema de búsqueda de vecinos que presenta DBSCAN para datos en tres dimensiones.

5. DBSCAN modificado

Considerando que uno de los principales cuellos de botella para el algoritmo es la búsqueda de vecinos, la modificación que se propone busca reducir tanto el tiempo de ejecución que le toma a ese paso, como las veces que se tiene que realizar. Para lograr el primer objetivo, se propone el uso de octrees debido a su eficiencia al indexar y buscar datos en tres dimensiones. Conseguir el segundo objetivo implica hacer una modificación a la forma en la que el algoritmo procesa los puntos. La idea general es la siguiente: Se determinan puntos núcleo con sus respectivos puntos alcanzables por densidad, pero, a diferencia del algoritmo original, no se expanden los grupos que se encuentran. Una vez que se encontraron

todos los grupos sin expandir, se calcula el centroide de estos, y se aplica el proceso de expansión a los centroides, añadiendo todos los puntos de los grupos que son alcanzables por densidad a un nuevo grupo. Esta modificación se explica más a detalle en las siguientes secciones.

5.1. Modificación con octrees

Se hizo una primera modificación a DBSCAN para reducir los tiempos de procesamiento al reemplazar el proceso de búsqueda de vecinos con distancia euclidiana por la búsqueda de vecinos con un octree. Para esto, antes de aplicar DBSCAN, es necesario generar un octree con la nube de puntos inicial. Una ventaja de los octrees es que se puede determinar que se termine la subdivisión del espacio cuando los nodos tengan una cierta longitud. Para el procesamiento de nubes de puntos, se eligió esa condición ya que se pueden tener varios puntos en un mismo nodo sin que posteriores búsquedas de vecinos se vean afectadas, y se logra que el octree resultante no tenga muchos nodos, reduciendo considerablemente el espacio de búsqueda. Otra ventaja de los octrees es que su tiempo de generación y de búsqueda se mantiene en $O(n \log n)$ [15]. Los resultados experimentales demuestran que el uso de octree reduce el tiempo de procesamiento del algoritmo.

5.2. Descripción de los dos pasos de agrupamiento

El algoritmo modificado requiere de dos pasos: búsqueda de puntos núcleo y generación de grupos iniciales, y de expansión de grupos basado en sus centroides. El primer paso es similar al algoritmo DBSCAN original, sin la parte de expansión de los grupos. Para este paso, el radio de búsqueda y los puntos mínimos son los mismos que para el algoritmo original. Es importante notar que para este proceso, sí se tienen que recorrer todos los puntos de la nube, sin embargo, se evita hacer búsquedas de región por cada uno de ellos. El resultado de este paso, que se puede ver en la Figura 3, es una lista que contiene los grupos de los puntos núcleo y los puntos que son alcanzables por densidad de estos. Hay que considerar que se va a tener una gran cantidad de grupos, pero en comparación con los puntos iniciales, el número es mucho menor: esto va a depender de el problema que se esté resolviendo, ya que el radio de búsqueda y los puntos mínimos afectan la cantidad de grupos que se obtienen. El pseudocódigo de este paso se puede ver en el Algoritmo 1.

El siguiente paso consiste en agrupar los grupos que se obtuvieron en el primer paso. Para esto, se obtienen los centroides de los grupos y se construye un segundo octree que será utilizado para hacer las siguientes búsquedas por región. El paso de expansión del algoritmo original se aplica a los centroides obtenidos, con el detalle de que las búsquedas van a ir agregando los puntos de cada uno de los grupos en lugar de los puntos de los centroides. Como se van a ir haciendo búsquedas por región sobre los centroides, el radio de búsqueda tiene que ser dos veces el radio de búsqueda original, para que a partir de cada uno hayan centroides que sean alcanzables por densidad. El resultado final, que

```
Octree = Crear_octree(minPts, nube_puntos)
Def DBSCAN_1(nube_puntos, Octree, eps, minPts):
grupos = lista()
for p en nube_puntos do
    if !visitado(p) then
        grupo = lista()
        marcar_visitado(p)
        grupo.agregar(p)
        vecinos = Consulta_vecinos_octree(Octree, p, eps)
        if vecinos < minPts then
            ruido.anadir(p)
        else
            for q en vecinos do
                if !visitado(q) then
                    marcar_visitado(q)
                    grupo.agregar(q)
                end
            end
            if tamano(grupo) > minPts then
                grupos.agregar(grupo)
            end
        end
    end
end
```

Algoritmo 1: Paso 1: Generación de grupos intermedios.

se puede ver en la Figura 4, son los grupos finales. En el Algoritmo 2 se puede ver el pseudocódigo de este paso.

6. Resultados experimentales

En esta sección se describe la metodología y resultados de los experimentos que fueron realizados para evaluar el algoritmo propuesto. El desarrollo de los algoritmos mencionados fue realizado en un equipo con el sistema operativo Windows 7, que cuenta con un procesador Intel Core i7-3612QM a 2.10GHz, en el ambiente de desarrollo Visual C++ 2013. Se hicieron pruebas sobre el siguiente conjunto de datos: un conjunto de 3600 nubes de puntos de profundidad que se grabó a 30 cuadros por segundo. Las nubes de puntos tienen entre 10000 y 46000 puntos de tres dimensiones, dependiendo de lo que se vea en cada cuadro de la escena. La escena que se grabó consiste de un fondo donde se observan manos y caras haciendo movimientos. Los datos se obtuvieron con el Kinect para Windows y el SDK 1.8.

Debido a que el objetivo es aplicar el algoritmo a una nube de puntos que se obtuvo con el Kinect, el radio de búsqueda que se usa como parámetro del algoritmo se estableció en 40 milímetros. Se eligió ese valor después de hacer un análisis del promedio de distancia entre cada punto y sus 4 vecinos

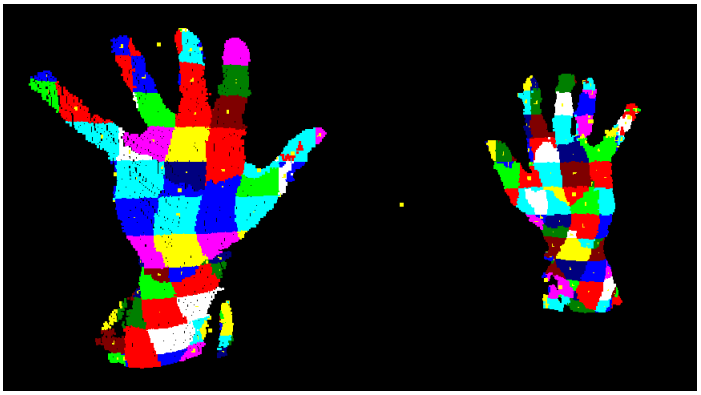

Fig. 3: Resultado de aplicar DBSCAN con un umbral de densidad menor a una nube de puntos. Los colores repetidos no indican que los puntos pertenecen al mismo grupo

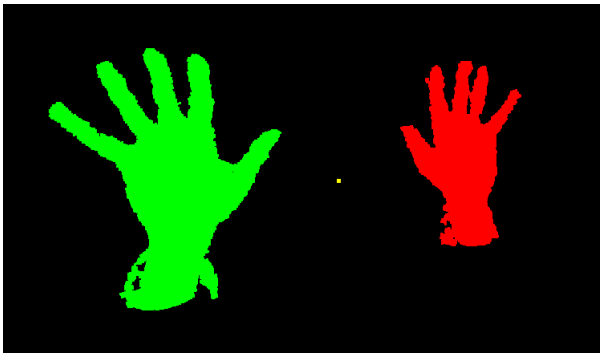

Fig. 4: Resultado de la expansión de los grupos obtenidos en el paso 1

más próximos [18]. El número mínimo de puntos para la primera parte del algoritmo se estableció en 100, mientras que para la segunda parte del algoritmo se estableció en 1000.

6.1. Desempeño del algoritmo

La Figura 5 muestra los promedios de tiempo de aplicar 100 veces los diferentes algoritmos a nubes de puntos con diferente número de puntos en cada una. En cada una de las pruebas se dieron los mismos resultados: El número de grupos fue el mismo, y se agruparon los mismos puntos, así como que se eliminaron los mismos puntos de ruido.

En primer lugar, se puede notar que utilizando el algoritmo original con octrees para realizar las búsquedas de vecinos, se obtiene una mejora considerable en tiempo en comparación con el algoritmo original. Esto se puede ver aún más claramente en la Figura 6, donde se puede ver el aumento de velocidad, en promedio de 970, que se tuvo al aplicar el algoritmo.

```
Centroides = lista()
for grupo en grupos do
    Centroides.agregar(Calcular_centroide(grupo))
end
Octree_2 = Crear_octree(minPts, Centroides)
Def DBSCAN_2(Centroides, grupos_iniciales, Octree_2, eps, minPts):
grupos_finales = lista()
for p en Centroides do
    if !visitado(p) then
        grupo = lista()
        marcar_visitado(p)
        grupo.agregar(Puntos(grupos_iniciales(p)))
        vecinos = Consulta_vecinos_octree(Octree_2, p, 2 * eps)
        for q en vecinos do
            if !visitado(q) then
                marcar_visitado(q)
                grupo.agregar(Puntos(grupos_iniciales(q)))
            end
        end
        if tamano(grupo) > minPts then
            grupos_finales.agregar(grupo)
        end
    end
end
```

Algoritmo 2: Paso 2: Expansión de grupos intermedios para obtener grupos finales.

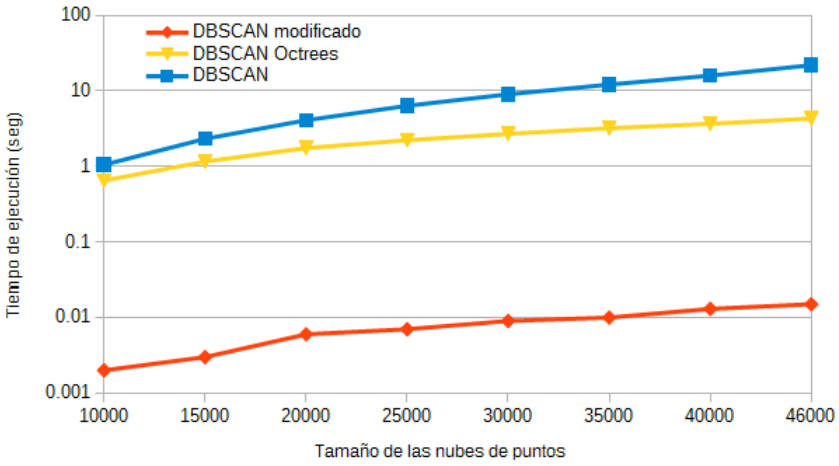

Fig. 5: Comparación de desempeño entre DBSCAN, DBSCAN solo con octrees para realizar las búsquedas, y el DBSCAN modificado

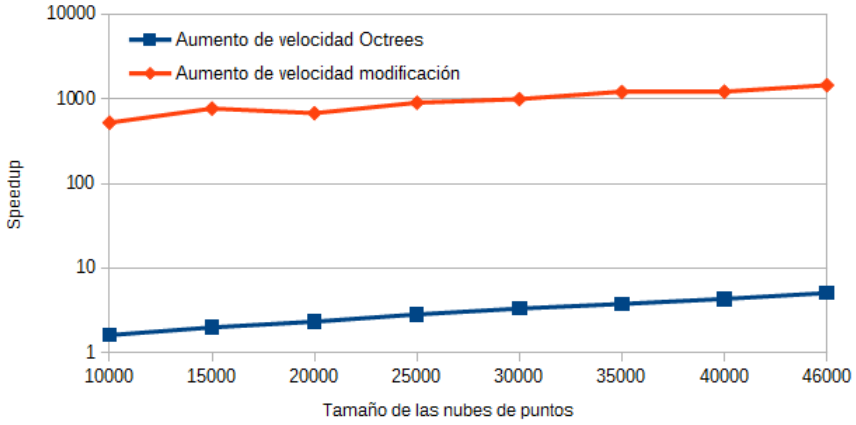

Fig. 6: Aumento de velocidad de los algoritmos

Al aplicar el método propuesto, se obtiene una mejora en tiempo que permite que se pueda usar en una aplicación interactiva, aún con nubes de puntos de gran tamaño. Con los tiempos que se obtuvieron para procesar una nube de puntos de 46000 puntos, se puede lograr un desempeño en tiempo de procesamiento de alrededor de 66 cuadros por segundo. Esto permite que se hagan otros procesos, y aún se pueda alcanzar el procesamiento en tiempo real, de alrededor de 30 cuadros por segundo.

7. Conclusión y trabajo a futuro

En este trabajo se presentó un algoritmo basado en DBSCAN para procesar nubes de puntos en tiempo real. Para lograr el procesamiento de las nubes en tiempo real, se tuvieron que utilizar tanto estructuras de datos espaciales, como son los octrees, como un esquema de particionamiento para reducir el espacio de búsqueda de vecinos. Se modificó el algoritmo original al separar en dos partes el procesamiento de los puntos. La primera parte genera grupos en base a puntos núcleo, y la segunda parte une esos grupos al aplicar la parte de expansión del algoritmo original. Todas las búsquedas de vecinos se realizan con los métodos de búsqueda por región que ofrecen los octrees. Al aplicar el algoritmo propuesto, y comparar los resultados de agrupamiento con el algoritmo original, se pudo observar que se obtenían los mismos grupos.

Como trabajo futuro se planea aplicar el algoritmo en un mayor número de conjuntos de datos, y no necesariamente datos de nubes de puntos obtenidas con algún sensor de profundidad. De igual manera, hay muchos algoritmos que han aprovechado arquitecturas en paralelo, tanto para CPU como para GPU, para acelerar el algoritmo DBSCAN. Se planea explorar dichos algoritmos para integrarlos con el algoritmo propuesto y mejorar los tiempos de procesamien-

to obtenidos. Específicamente, se plantea el uso de GPU para poder hacer el procesamiento de nubes de puntos de mayor tamaño al utilizado en éste trabajo.

Referencias

1. Arlia, D., Coppola, M.: Experiments in parallel clustering with dbscan. In: Euro-Par 2001 Parallel Processing, pp. 326–331. Springer (2001)
2. Bianchi, L., Martinelli, A.: A clustering approach to object estimation, featuring image filtering prototyping for dbscan in virtual sets. In: Image Analysis and Processing, 2007. ICIAP 2007. 14th International Conference on. pp. 751–756. IEEE (2007)
3. Borah, B., Bhattacharyya, D.: An improved sampling-based dbscan for large spatial databases. In: Intelligent Sensing and Information Processing, 2004. Proceedings of International Conference on. pp. 92–96. IEEE (2004)
4. Doan, N.H., Pham, D., Dinh, T.B., Dinh, T.B.: Restoring surfaces after removing objects in indoor 3d point clouds. In: Proceedings of the Fourth Symposium on Information and Communication Technology. pp. 189–197. ACM (2013)
5. Ester, M., Kriegel, H.P., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: International Conference on Knowledge Discovering in Databases and Data Mining. vol. 96, pp. 226–231 (1996)
6. Gan, J., Tao, Y.: Dbscan revisited: Mis-claim, un-fixability, and approximation. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. pp. 519–530. ACM (2015)
7. Ghosh, S., Lohani, B.: Development and comparison of aerial photograph aided visualization pipelines for lidar datasets. International Journal of Digital Earth 8(8), 656–677 (2015)
8. Götz, M., Bodenstein, C., Riedel, M.: Hpdbscan: highly parallel dbscan. In: Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments. p. 2. ACM (2015)
9. Gunawan, A., de Berg, M.: A faster algorithm for DBSCAN. Ph.D. thesis, Master (2013)
10. Joshi, D., Samal, A.K., Soh, L.K.: Density-based clustering of polygons. In: Computational Intelligence and Data Mining, 2009. CIDM'09. IEEE Symposium on. pp. 171–178. IEEE (2009)
11. Liu, B.: A fast density-based clustering algorithm for large databases. In: Machine Learning and Cybernetics, 2006 International Conference on. pp. 996–1000. IEEE (2006)
12. Lun, R., Zhao, W.: A survey of applications and human motion recognition with microsoft kinect. International Journal of Pattern Recognition and Artificial Intelligence 29(05), 1555008 (2015)
13. Meagher, D.J.: Octree encoding: A new technique for the representation, manipulation and display of arbitrary 3-d objects by computer. Electrical and Systems Engineering Department Rensseiaer Polytechnic Institute Image Processing Laboratory (1980)
14. Microsoft: Kinect for windows programming guide. https://msdn.microsoft.com/en-us/library/hh855348.aspx, accesado: 2015-04-11
15. Samet, H.: Spatial Data Structures: Quadtree, Octrees and Other Hierarchical Methods. Addison Wesley Reading (1989)

16. Shotton, J., Sharp, T., Kipman, A., Fitzgibbon, A., Finocchio, M., Blake, A., Cook, M., Moore, R.: Real-time human pose recognition in parts from single depth images. Communications of the ACM 56(1), 116–124 (2013)
17. Thapa, R.J., Trefftz, C., Wolffe, G.: Memory-efficient implementation of a graphics processor-based cluster detection algorithm for large spatial databases. In: Electro/Information Technology (EIT), 2010 IEEE International Conference on. pp. 1–5. IEEE (2010)
18. Zhou, A., Zhou, S., Cao, J., Fan, Y., Hu, Y.: Approaches for scaling dbscan algorithm to large spatial databases. Journal of computer science and technology 15(6), 509–526 (2000)
19. Zhou, X.H., Wang, H.B., Zhou, D.R., Meng, B.: Data clustering algorithm based on digital search tree. In: Machine Learning and Cybernetics, 2003 International Conference on. vol. 3, pp. 1757–1761. IEEE (2003)