Journal of the Korean Society of Surveying, Geodesy, Photogrammetry and Cartography
Vol. 32, No. 5, 481-486, 2014
http://dx.doi.org/10.7848/ksgpc.2014.32.5.481

ISSN 1598-4850(Print)
ISSN 2288-260X(Online)
Original article

등축형 공간 분할과 독립적 옥트리 생성을 통한 대용량 3차원 포인트 클라우드의 탐색 효율 향상

Enhancing Query Efficiency for Huge 3D Point Clouds Based on Isometric Spatial Partitioning and Independent Octree Generation

한수희[1)]
Han, Soohee

Abstract

This study aims at enhancing the performance of file-referring octree, suggested by Han(2014), for efficiently querying huge 3D point clouds, acquired by the 3D terrestrial laser scanning. Han's method(2014) has revealed a problem of heavy declining in query speed, when if it was applied on a very long tunnel, which is the lengthy and narrow shaped anisometric structure. Hereupon, the shape of octree has been analyzed of its influence on the query efficiency with the testing method of generating an independent octree in each isometric subdivision of 3D object boundary. This method tested query speed and main memory usage against the conventional single octree method by capturing about 300 million points in a very long tunnel. Finally, the testing method resulted in which twice faster query speed is taking similar size of memory. It is also approved that the conclusive factor influencing the query speed is the destination level, but the query speed can still increase with more proximity to isometric bounding shape of octree. While an excessive unbalance of octree shape along each axis can heavily degrade the query speed, the improvement of octree shape can be more effectively enhancing the query speed than increasement of destination level.

Keywords : LiDAR, 3D Point Cloud, Query, Octree, File-referring, Octree Shape

초 록

본 연구는 3차원 지상레이저스캐닝을 통해 취득된 대용량 3차원 포인트 클라우드를 효율적으로 탐색하기 위하여 Han(2014)이 제안한 파일 참조 옥트리의 성능을 개선하는 것을 목표로 한다. Han(2014)의 방식을 좁고 긴 형상의 비등축형 구조물인 장대터널에 적용한 결과 포인트 탐색 속도가 크게 저하되는 문제점이 발견되었다. 이에 옥트리의 형상이 포인트 클라우드의 탐색 효율에 미치는 영향을 분석하였으며, 대상물의 3차원 경계를 등축형 하위 영역으로 분할하고 각 영역에 독립적인 옥트리를 생성하는 방식을 제안하였다. 장대터널에서 취득된 약 3억 개의 포인트로부터 단일 옥트리를 생성하는 기존 방식과 다수의 독립적인 옥트리를 생성하는 방식으로 포인트 탐색 속도와 메인 메모리 사용량을 비교하였다. 결과로 다수 옥트리 방식이 유사한 크기의 메인 메모리를 사용면서도 약 2배의 탐색 속도를 나타내었다. 아울러 옥트리의 탐색 속도를 좌우하는 주요 요소는 목표 단계이나 같은 목표 단계에서는 옥트리의 형상이 등축형에 가까울수록 탐색 속도는 여전히 증가함을 확인하였다. 다만 옥트리의 형상이 각 축 방향으로 지나치게 불균형을 이룰 경우 탐색 속도는 크게 저하되며 이 경우 옥트리의 형상 개선이 목표 단계 증가보다 탐색 속도 향상에 효과가 큼을 확인하였다.

핵심어 : 라이다, 3차원 포인트 클라우드, 탐색, 옥트리, 파일참조, 옥트리 형상

Received 2014. 10. 04, Revised 2014. 10. 20, Accepted 2014. 10. 21
1) Member, Dept. of Geoinformatics Engineering, Kyungil University (E-mail:scivile@kiu.ac.kr)

1. 서 론

차원 지상레이저스캐닝은 대규모 건설현장에서 토탈스테이션을 이용한 재래식 측량을 대체할 수 있는 첨단 측량 기술로 인식되고 있다. 특히 터널이나 교량, 댐과 같이 국소 표면의 변형이나 부재의 거동이 전체 구조물의 안전에 영향을 미칠 수 있는 경우 지상레이저스캐닝을 이용하여 광범위한 영역에 대해 높은 밀도의 측량을 효율적으로 수행할 수 있다. 한편 스캐닝 장비의 발달과 저장 장치의 가격 하락에 힘입어 지상레이저스캐닝을 통해 취득된 포인트 클라우드의 용량이 급격히 증가하고 있는 가운데 이를 효율적으로 활용하기 위해서는 적절한 색인(indexing) 방식의 사용이 필수적이다(Han *et al*., 2011). 옥트리는 이미 2.5차원 및 3차원 포인트 클라우드의 처리와 관련된 다양한 연구에서 사용되고 있으며(Saxena *et al*., 1995; Woo *et al*., 2002; Wang and Tseng, 2004; Schnabel *et al*., 2007; Cho *et al*., 2008; Marechal, 2009), 메모리 사용량 및 탐색 속도 면에서 합리적인 방식임이 증명되었다(Han *et al*., 2011; Han *et al*., 2012). Han(2013)은 옥트리 생성에 소요되는 메인 메모리의 사용량을 줄이기 위하여 C언어 환경에서 트리를 구성하는 노드 클래스의 멤버 변수와 하위 노드 검색 방식이 미치는 영향을 분석하고 개선 방안을 제시하였다. 그러나 Han(2013)의 방식은 3차원 포인트 클라우드 전체를 메인 메모리에 저장하여 탐색하는 방식으로서 메인 메모리의 용량을 초과하는 포인트 클라우드에 대해서는 적용할 수 없다는 단점이 있다. 이에 Han(2014)은 포인트 클라우드를 메인 메모리에 저장하지 않고 하드디스크에 저장한 상태로 각 포인트에 대한 파일 포인터를 직접적으로 참조하는 방법을 사용함으로써 색인 가능한 포인트 클라우드의 용량을 증가시켰다. 아울러 메인 메모리에 구현된 옥트리를 파일로 저장하고 필요시 복원함으로써 옥트리 구성 시간을 현저하게 줄이는 방법도 제안하였다.

한편 Han(2014)의 방식을 좁고 긴 형상의 구조물인 장대터널에 적용한 결과 포인트 탐색 속도가 크게 저하되는 문제점이 발견되었다. 일반적으로 옥트리는 정육면체의 형상으로 구현하나 대상물을 둘러싸는 최소 크기의 직육면체로 구현하는 것이 리프 노드의 포인트 밀도를 보다 균일하게 유지하여 탐색 효율 제고에 유리하다. 그러나 장대터널과 같이 종방향으로 현저히 긴 대상물을 둘러싸는 형상의 옥트리를 구현하면 개별 리프 노드가 감당하는 포인트의 수가 늘어 탐색 효율이 저하된다. 이에 본 연구에서는 옥트리의 형상이 포인트 클라우드의 탐색 효율에 미치는 영향을 분석하고 터널과 같이 x, y, z 방향으로 현저한 크기 차이를 나타내는 대상물의 포인트 클라우드를 대상으로 탐색 효율을 제고할 수 있는 옥트리 구성 방안을 제시하고자 한다.

2. 본 론

기존의 연구에서는 3차원 포인트 클라우드가 차지하는 전체 영역을 둘러싸는 최소의 직육면체에 대하여 단일 옥트리를 생성하였다. 직육면체의 좌하단 및 우상단 3차원 좌표를 각각 $(x_{\min}, y_{\min}, z_{\min}), (x_{\max}, y_{\max}, z_{\max})$라 하고 x, y, z 방향의 길이를 각각 $l_x^0(=x_{\max}-x_{\min})$, $l_y^0(=y_{\max}-y_{\min})$, $l_z^0(=z_{\max}-z_{\min})$라 하면 2번 분할된 상태, 즉 2단계의 목표 단계에서의 옥트리의 형상은 Fig. 1과 같고 n단계 목표 단계에서 리프노드의 크기는 Eq. (1)과 같이 정의한다.

$$l_x^n = l_x^0/2^n,\ l_y^n = l_y^0/2^n,\ l_z^n = l_z^0/2^n \tag{1}$$

아울러 리프노드의 각 축간 크기의 비는 전체 영역을 둘러싸는 직육면체의 각 축간 비율과 같다(Eq. (2)).

$$l_{\mathrm{x}}^n : l_{\mathrm{y}}^n : l_{\mathrm{z}}^n = l_{\mathrm{x}}^0 : l_{\mathrm{y}}^0 : l_{\mathrm{z}}^0 \tag{2}$$

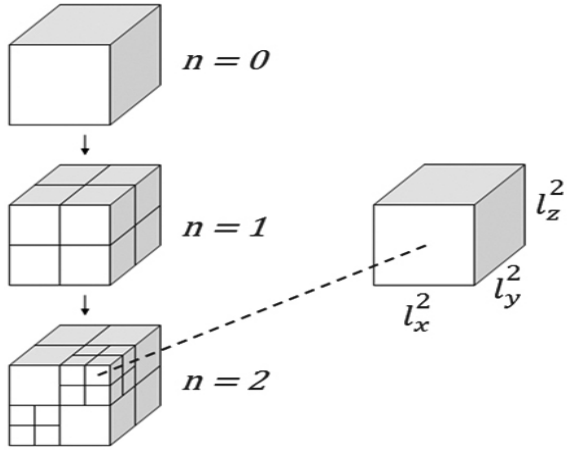

Fig. 1. Octree in destination level 2

만일 장대 터널 또는 교량과 같이 전체 영역을 둘러싸는 직육면체의 수평 방향 길이가 수직 방향 길이에 비하여 매우 커서 불균형을 이루는 경우 리프노드의 수평 및 수직 방향 크기 비율 역시 불균형을 이루게 된다. 이 때 Han(2014)의 연구처럼 일정 반경 안에 존재하는 포인트를 탐색하는 경우 필요 이상으로 많은 인근 포인트에 대한 탐색을 수행해야 한다. 예를 들어 Fig. 2와 같이 포인트 클라우드의 분포 형상이 z축 방향에

비하여 x축 방향으로 긴 경우 단일 옥트리를 생성하면 리프노드 역시 Fig. 3(a)와 같이 x축 방향으로 긴 형상을 갖게 된다. 이 경우 중심으로부터 일정 거리에 있는 포인트를 탐색하기 위해 접근해야 하는 리프노드의 수는 8개이지만 그 안에 속하여 접근해야 하는 포인트의 수는 29개이다. 만일 Fig. 3(b)와 같이 리프노드의 형상이 등축형에 가까우면 12개의 리프노드에 속한 17개의 포인트에만 접근하면 된다. 즉 리프노드의 형상이 등축형에 가까울수록 접근해야 하는 포인트가 수가 감소하므로 탐색 속도가 빨라진다. 특히 Han(2014)의 연구와 같이 옥트리는 접근 속도가 빠른 메인 메모리에 형성되어 있고 리프노드에 속한 포인트는 상대적으로 매우 느린 하드디스크드라이브를 참조할 경우 리프노드의 형상 변화에 의한 효과는 더욱 커질 것으로 기대할 수 있다.

이에 본 연구에서는 Fig. 4와 같이 포인트 클라우드 전체를 둘러싸는 최소 직육면체를 가급적 등축형에 가까운 동일한 크기의 하위 공간으로 분할하고 각각의 하위 공간에 대하여 독립적인 옥트리를 생성하는 방식을 제안한다. 다만 다수의 옥트리를 생성할 경우 메모리의 사용량이 증가한다는 단점이 존재한다. 예를 들어 Fig. 4처럼 4개의 옥트리를 독립적으로 생성하면 Fig. 2의 단일 옥트리에 비해 최대 4배의 메모리를 소요할 수 있다. 따라서 하위 공간의 형상이 등축형에 가까운 정도에 따라 귀결되는 탐색 속도와 메인 메모리 사용량 등을 분석하고자 한다. 이를 위해 분할 전 직육면체의 x, y, z 방향 길이 중 최단 길이와 분할 후 공간의 x, y, z 방향 길이 중 어느 것과의 비율도 임계치 t_l (1 이상의 실수)을 초과할 수 없도록 공간을 분할한다. 분할 전 공간의 x, y, z 방향 길이 l_x^0, l_y^0, l_z^0 를 크기 순서대로 l_1^0, l_2^0, $l_3^0 (l_1^0 \geq l_2^0 \gg l_3^0)$로 정의하면, l_1^0과 l_2^0방향으로 분할될 공간의 수 n_1과 n_2는 Eq. (3)과 같이 정의한다.

$$n_1 = \mathrm{ceil}\left(\frac{l_0^1}{l_0^3 \times t_l}\right),\ n_2 = \mathrm{ceil}\left(\frac{l_0^2}{l_0^3 \times t_l}\right) \tag{3}$$

Eq. (3)에서 ceil(*r*)은 실수 *r*보다 작지 않은 최소의 정수로 정의한다. 분할된 공간의 l_1^0, 과 l_2^0 방향 길이 $l_1^{0\prime}$ 과 $l_2^{0\prime}$ 는 식 (4)와 같이 정의한다.

$$l_1^{0\prime} = l_1^0/n_1,\ l_2^{0\prime} = l_2^0/n_2 \tag{4}$$

예를 들어 $t_l = 1$ 인 경우 $l_1^{0\prime}$과 $l_2^{0\prime}$ 의 크기는 l_3^0 보다 크지 않으며 분할된 공간은 거의 등축형을 이루게 된다. $t_l = 2$인 경우 $l_1^{0\prime}$과 $l_2^{0\prime}$의 크기는 l_3^0 의 두 배보다 크지 않으며 분할된 공간은 두 축이 나머지 한 축에 비해 거의 2배인 직육면체의 형상을 이루게 된다.

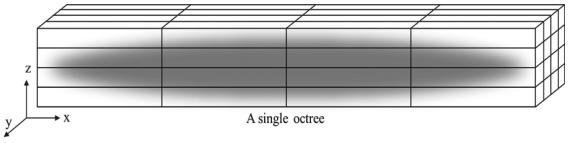

Fig. 2. Generation of a single anisometric octree

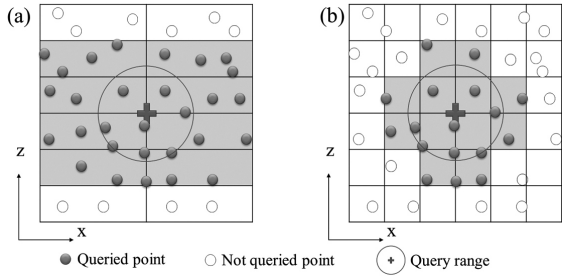

Fig. 3. Comparison of query overheads: (a) anisometric leaf nodes, (b) near-isometric leaf nodes

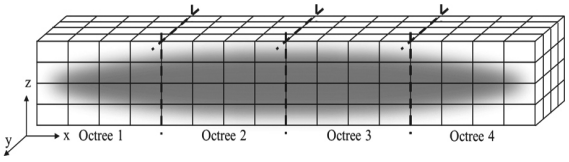

Fig. 4. Generation of multiple isometric octrees

3. 적용 및 평가

본 연구에서는 3차원 포인트 클라우드로부터 옥트리를 생성하고 저장하며 재현하기 위하여 소요된 시간과 메인 메모리의 크기를 측정하였으며, 옥트리의 탐색 성능을 파악하기 위하여 일정 반경 안에 존재하는 인근 3차원 포인트에 대한 탐색 시간을 측정하였다. 비교 대상은 Han(2014)의 파일 참조 방식과 동일한 단일 옥트리를 생성하는 방식(이 후 단일 옥트리 방식)과 본 연구에서 제안한 다수의 독립적인 옥트리를 생성하는 방식(이 후 다수 옥트리 방식)이다. 실험에 사용한 포인트 클라우드는 길이가 약 1.5km인 철도 터널에 대한 3차원 지상레이저스캐닝으로 취득된 자료이며 보안상 정확한 위치와 출처는 생략한다. 포인트 클라우드의 제원은 Table 1과 같으며 실험에 사용된 시스템의 사양은 Table 2와 같다. 포인트 클라우드의 x 방향 크기는 569.16m, y 방향 크기는 1442.58m 임에 비하여 z 방향 크기는 19.05m로서 수평 방향과 연직 방향이 큰 차이를 보이고 있다. Han(2014)의 연구에서는 포인트

Table 1. Specifications of point cloud

Terrestrial laser scanner	C10, Leica Geosystems
Scanned object	A train tunnel of 1.5km length $l_x^0 = 569.16$m, $l_y^0 = 1442.58$m, $l_z^0 = 19.05$m
Number of points	300525406
File size	6879 MB
File structure	x, y, z (3 doubles in binary format)

Table 2. Specifications of system

CPU	AMD FX 8150 (8 cores at 3.6GHz)
RAM	8 GB
OS	Windows server 2008 (64bit)
Compiler	C++ 64bit release, Visual Studio 2010
SSD	120 GB

클라우드를 저장하고 파일 포인터를 참조하기 위하여 하드디스크드라이브를 사용하였으나 본 실험에서는 보다 안정적인 성능 평가를 위하여 SSD(Solid State Drive)를 사용하였다.

단일 옥트리 방식의 경우 공간 분할 횟수인 목표 단계(Han, 2013)를 9부터 13까지 설정하였다. 목표 단계 14부터는 시스템이 기본적으로 사용하는 백그라운드 서비스와 함께 8GB 이상의 메인 메모리를 소요하여 시스템이 페이지 메모리를 사용함으로써 정확한 성능 측정이 불가함에 따라 실험에서 제외하였다. 다수 옥트리 방식의 목표 단계는 8부터 10까지 설정하였으며 분할된 공간의 축간 길이 비율 임계치인 t_l 은 1부터 3의 정수로 설정하였다. 아울러 목표 단계 10에서 $t_l = 1$인 경우 백그라운드 서비스와 함께 8GB 이상의 메인 메모리를 소요하여 실험에서 제외하였다.

각 목표 단계에서 리프 노드의 크기와 옥트리의 생성, 저장, 재현에 소요된 시간 및 메모리의 크기는 Table 3과 같다. 단일 옥트리 방식의 경우 리프노드의 최장 축(y축)과 최단 축(z축)의 크기 비율이 약 60배의 불균형을 이룬다. 반면 다수 옥트리 방식의 경우는 최장 축(y축)과 최단 축(z축)의 크기 비율이 $t_l = 1$일 때 1배, $t_l = 2$일 때 2배, $t_l = 3$일 때 3배 미만으로 불균형이 크게 해소된 것을 확인할 수 있다. 옥트리의 생성, 저장, 재현에 소요된 시간은 옥트리 구현에 소요된 메인 메모리의 크기와 밀접한 관련이 있는 것으로 파악되었으며 전반적으로 큰 차이를 보이지 않았다.

옥트리의 탐색 성능을 평가하기 전체 포인트 수의 약 0.01%인 30054개의 포인트를 중심으로 반경 5cm 이내의 포인트를 탐색하였다. 모든 경우에 대하여 탐색된 포인트의 수는 17554511개로 동일하여 옥트리의 탐색 정확성에는 결함이 없는 것을 확인할 수 있다. 탐색 시간과 사용된 메인 메모리의 크기는 Table 4와 같으며 탐색 속도에 따라 오름차순으로 정렬하였다.

Table 4의 case 1에서 다수 옥트리 방식의 목표 단계를 10, $t_l = 2$로 설정할 때 가장 빠른 속도를 기록하였고 단일 옥트리 방식에서 목표 단계를 12, 13으로 설정하였을 때를 제외하

Table 3. Time and main memory taken by octree generation

Method	Level (n)	t_l	l_x^n(cm)	l_y^n(cm)	l_z^n(cm)	Building (sec)	Saving (sec)	Restoring (sec)	Memory (MB)
A single octree	9	n/a	111.17	223.16	3.72	1524.46	25.73	32.07	2663
	10	n/a	55.58	111.58	1.86	1593.74	25.37	30.20	2849
	11	n/a	27.79	55.79	0.93	1775.48	26.74	33.52	3174
	12	n/a	13.90	27.89	0.47	1910.86	31.73	43.57	4083
	13	n/a	6.95	13.95	0.23	2007.52	50.14	67.94	6416
Multiple octrees	8	3	22.23	22.32	7.44	1336.18	26.60	28.74	2948
	8	2	14.82	14.88	7.44	1764.00	25.66	35.02	3056
	8	1	7.41	7.44	7.44	1783.79	26.82	34.79	3379
	9	3	11.12	11.16	3.72	1778.01	26.80	37.08	3311
	9	2	7.41	7.44	3.72	1819.70	28.35	38.16	3585
	9	1	3.71	3.72	3.72	1856.63	33.99	54.49	4484
	10	3	5.56	5.58	1.86	1879.56	33.21	45.19	4362
	10	2	3.71	3.72	1.86	1904.65	38.22	55.96	5111

고는 다수 옥트리 방식의 탐색 속도가 우세했다. 다수 옥트리 방식과 단일 옥트리 방식에서 각각 가장 빠른 속도를 비교하면 case 1과 case 3에서 1/366.99sec : 1/582.60sec ≒ 1.59 : 1로서 다수 옥트리 방식이 약 1.6배 빠른 것을 확인할 수 있다. 반면 당시의 메인 메모리 사용량은 5111MB : 6416MB ≒ 0.80 : 1로서 다수 옥트리 방식이 20% 가량 적게 사용함을 확인할 수 있다. 아울러 유사한 크기의 메인 메모리를 사용할 경우 다수 옥트리 방식이 단일 옥트리 방식에 비해 탐색 속도가 빠름을 확인할 수 있다. 예를 들어 case 12와 case 10에서 다중 옥트리 방식이 단일 옥트리 방식에 비하여 메인 메모리를 3% 가량 많이 소요하지만 탐색 속도는 2.8배 빠름을 확인할 수 있으며, case 9와 case 11에서 메모리는 4% 가량 적게 소요하면서 탐색 속도는 1.6배 빠름을 확인할 수 있다.

한편 다수 옥트리 방식의 경우 t_l 이 1에 가까울수록, 즉, 옥트리의 형상이 등축형에 가까울수록 탐색 속도가 명백히 빨라짐을 확인할 수 있다. 다만 이에 의한 효과는 목표 단계의 증가로 인한 속도 향상 효과를 능가할 수는 없음을 확인할 수 있다. 예를 들어 목표 단계 9에서 t_l 값에 관계 없이 목표 단계 8보다 탐색 속도가 빠르며 목표 단계 10 역시 목표 단계 9보다 빠르다.

Table 4. Comparisons of main memory usage and query time

Case	Method	Level	t_l	Memory (MB)	Query (sec)
1	Multiple	10	2	5111	366.99
2	Multiple	10	3	4362	383.47
3	Multiple	9	1	4484	416.10
4	Single	13	n/a	6416	582.60
5	Multiple	9	2	3585	582.77
6	Multiple	8	1	3379	601.01
7	Multiple	9	3	3311	878.05
8	Single	12	n/a	4083	1071.13
9	Multiple	8	2	3056	1173.55
10	Multiple	8	3	2948	1261.58
11	Single	11	n/a	3174	1856.79
12	Single	10	n/a	2849	3537.23
13	Single	9	n/a	2663	8389.89

Fig. 5는 소요된 메모리의 크기와 탐색 시간의 추세를 그래프로 나타낸 것으로 소요되는 메인 메모리의 크기가 증가할수록 탐색 속도는 빨라진다는 일반적인 원리를 확인할 수 있다. 아울러 다중 옥트리 방식이 단일 옥트리 방식과 비교하여 비슷한 크기의 메인 메모리를 소요할 경우 탐색 속도는 2배 가량 빠름을 확인할 수 있다. 따라서 본 연구에서 제안한 다중 옥트리 방식이 보다 단일 옥트리 방식에 비하여 성능이 우수하다고 결론지을 수 있다.

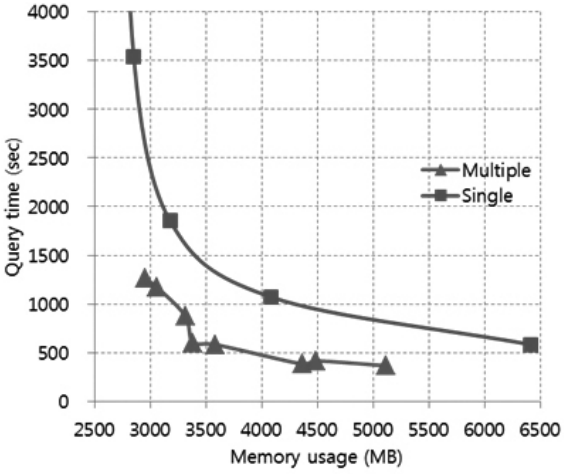

Fig. 4. Generation of multiple isometric octrees

4. 결 론

본 연구에서는 3차원 지상레이저스캐너로부터 취득한 대용량의 포인트 클라우드로부터 효율적인 탐색을 수행하기 위하여 Han(2014)이 제안한 파일 참조 옥트리의 성능을 향상시키고자 하였다. 특히 장대터널과 같이 연직 방향에 비해 종방향의 크기가 매우 큰 대상물에 대하여 취득한 포인트 클라우드로부터 옥트리를 생성할 경우 리프 노드의 형상이 특정 방향으로 지나치게 커져 탐색 성능이 저하되는 문제점이 발생할 수 있다. 이를 해결하기 위해 본 연구에서는 포인트 클라우드를 감싸는 3차원 공간을 가급적 등축형에 가까운 하위 공간으로 분할하고 각 공간에 대하여 독립적인 옥트리를 생성하는 방법을 제안하였다. 본 연구를 통하여 다음과 같은 결론을 내릴 수 있다.

- 일반적으로 옥트리의 탐색 속도를 증가시키려면 메인 메모리의 추가적인 소요를 감수해야 한다.
- 옥트리의 탐색 속도를 좌우하는 주요 요소는 목표 단계이며 같은 목표 단계에서는 옥트리의 형상이 등축형에 가까울수록 탐색 속도는 증가한다.
- 옥트리의 형상이 각 축 방향으로 지나치게 불균형을 이룰 경우 탐색 속도는 크게 저하되며 이 경우 옥트리의 형상 개선이 목표 단계 증가보다 탐색 속도 향상에 효과가 크다.

Han(2014)의 연구와 본 연구에서는 메인 메모리에 옥트리를 생성하고 파일 포인터를 리프노드에 저장하는 방식을 사용함으로써 옥트리 생성에 여전히 많은 메모리가 소요되며 잦은 파일 참조로 인해 탐색 속도 향상에는 한계가 있다. 향후 연구에서는 파일 포인터 저장과 잦은 파일 참조를 해소할 수 있는 방법을 강구하고자 한다.

References

Cho, H., Cho, W., Park, J., and Song, N. (2008), 3D building modeling using aerial LiDAR data, *Korean Journal of Remote Sensing*, Vol. 24, pp. 141–152. (in Korean with English abstract)

Marechal, L. (2009), Advances in octree-based all-hexahedral mesh generation: handling sharp features, *Proceedings of 18th International Meshing Roundtable*, Salt Lake City, UT, USA, pp. 65–84.

Han, S., Lee, S., Kim, S. P., Kim, C., Heo, J., and Lee, H. (2011), A comparison of 3D R-tree and octree to index large point clouds from a 3D terrestrial laser scanner, *Journal of the Korean Society of Surveying, Geodesy, Photogrammetry and Cartography*, Vol. 29, No. 1, pp. 531-537. (in Korean with English abstract)

Han, S., Kim, S., Jung, J. H., Kim, C., Yu, K., and Heo, J. (2012), Development of a hashing-based data structure for the fast retrieval of 3D terrestrial laser scanned data, *Computers & Geosciences*, Vol. 39, pp. 1-10.

Han, S. (2013), Design of memory-efficient octree to query large 3D point cloud, *Journal of the Korean Society of Surveying, Geodesy, Photogrammetry and Cartography*, Vol. 31, No. 1, pp. 41-48. (in Korean with English abstract)

Han, S. (2014), Implementation of file-referring octree for huge 3D point clouds, *Journal of the Korean Society of Surveying, Geodesy, Photogrammetry and Cartography*, Vol. 32, No. 2, pp. 109-115. (in Korean with English abstract)

Saxena, M., Finnigan, P. M., Graichen, C. M., Hathaway, A. F., and Parthasarathy, V. N. (1995), Octree-based automatic mesh generation for non-manifold domains, *Engineering with Computers*, Vol. 11, pp. 1–14.

Schnabel, R., Wahl, R., and Klein, R. (2007), Efficient RANSAC for point-cloud shape detection, *Computer Graphics Forum*, Vol. 26, pp. 214–226.

Wang, M. and Tseng, Y.-H. (2004), Lidar data segmentation and classification based on octree structure, *Proceedings of XXth ISPRS Congress*, ISPRS, Istanbul, Turkey.

Woo, H. ,Kang, E., Wang, S., and Lee, K. H. (2002), A new segmentation method for point cloud data. *International Journal of Machine Tools and Manufacture*, Vol. 42, pp. 167–178.