Journal of the Korean Society of Surveying, Geodesy, Photogrammetry and Cartography
Vol. 33, No. 5, 437-442, 2015
http://dx.doi.org/10.7848/ksgpc.2015.33.5.437

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

포인트 클라우드 파일의 측점 재배치를 통한 파일 참조 옥트리의 성능 향상

Improving Performance of File-referring Octree Based on Point Reallocation of Point Cloud File

한수희[1)]
Han, Soohee

Abstract

Recently, the size of point cloud is increasing rapidly with the high advancement of 3D terrestrial laser scanners. The study aimed for improving a file-referring octree, introduced in the preceding study, which had been intended to generate an octree and to query points from a large point cloud, gathered by 3D terrestrial laser scanners. To the end, every leaf node of the octree was designed to store only one file-pointer of its first point. Also, the point cloud file was re-constructed to store points sequentially, which belongs to a same leaf node. An octree was generated from a point cloud, composed of about 300 million points, while time was measured during querying proximate points within a given distance with series of points. Consequently, the present method performed better than the preceding one from every aspect of generating, storing and restoring octree, so as querying points and memorizing usage. In fact, the query speed increased by 2 times, and the memory efficiency by 4 times. Therefore, this method has explicitly improved from the preceding one. It also can be concluded in that an octree can be generated, as points can be queried from a huge point cloud, of which larger than the main memory.

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

초 록

최근 3차원 지상 레이저 스캐너의 성능이 고도로 향상됨에 따라 취득된 측점들로 구성된 포인트 클라우드의 용량도 급격히 증가하고 있다. 본 연구는 3차원 지상 레이저 스캐너로부터 취득한 대용량 포인트 클라우드로부터 옥트리를 생성하고 측점을 질의하기 위한 선행 연구의 파일 참조 옥트리 방식을 개선하는 것을 목표로 한다. 이를 위하여 메인 메모리에 구현된 옥트리의 리프 노드에는 첫 번째 측점의 파일 포인터만을 저장하였다. 아울러 동일한 리프 노드에 속하는 측점들이 연속적으로 기록되도록 포인트 클라우드 파일을 재구성하였다. 약 3억 개의 측점으로 구성된 포인트 클라우드로부터 옥트리를 생성하고 일련의 측점 주위로 일정 반경 안에 존재하는 측점들에 대한 질의 시간을 측정하였다. 결과적으로 옥트리의 생성 시간, 저장과 복원 시간, 질의 시간 및 메모리 사용량 등 모든 면에서 제안한 방식이 기존 방식에 비하여 향상된 성능을 나타내었다. 특히 질의 속도는 2배 이상, 메모리 효율성은 4배 이상 증가하였다. 따라서 본 연구는 선행 연구의 방식을 명백히 향상시켰다고 판단할 수 있다. 아울러 메인 메모리의 크기를 크게 상회하는 초대용량 포인트 클라우드로부터 옥트리를 구성하고 측점을 질의하는 것이 가능할 것으로 판단된다.

핵심어 : 라이다, 3차원 포인트 클라우드, 질의, 옥트리, 파일참조

Received 2015. 10. 06, Revised 2015. 10. 24, Accepted 2015. 10. 24
1) Member, Dept. of Geoinformatics Engineering, Kyungil University (E-mail: scivile@kiu.ac.kr)

1. 서 론

최근 3차원 지상 레이저 스캐너의 성능이 고도로 향상됨에 따라 취득된 측점들로 구성된 포인트 클라우드(point cloud)의 용량도 급격히 증가하고 있다. 기종에 따라 차이는 있지만 3차원 지상 레이저 스캐너 중 RIEGL VZ-2000 제품은 초당 40만개의 측점을 취득할 수 있고(RIEGL, 2015a), 이동 장치에 장착 가능한 2차원 레이저스캐너 중 RIEGL VUX-1HA 제품은 초당 100만개의 측점을 취득할 수 있는 것으로 알려져 있다(RIEGL, 2015b). 초당 40만개의 측점 취득을 가정할 경우 10분간 레이저스캐닝을 수행하면 2억 4000만개의 측점이 취득되며 측점의 3차원 좌표 x, y, z를 4byte 단정밀도 실수(float)로 저장할 경우 약 5.36GB의 용량을 차지하게 된다. HDD와 SDD등 고속 저장 장치의 용량 확장과 가격 하락에 힘입어 이와 같은 대용량의 측점을 저장하는 것은 가능하나, 임의의 위치에 존재하는 측점을 질의(query)하는 것은 결코 쉬운 일이 아니다. 왜냐하면 측점이 저장된 파일상 주소와 측점의 공간상 좌표는 상관성이 희박하기 때문이다. 따라서 특정 좌표 또는 그 인근에 존재하는 한 개의 측점을 질의하기 위해서는 최악의 경우 brute-force search(Wikipedia, 2015), 즉, 파일의 시작부터 끝까지의 검색해야 하는 경우도 발생할 수 있다. 따라서 3차원 지상 레이저 스캐너로부터 취득한 포인트 클라우드의 색인(indexing)을 위한 적절한 구조체의 도입이 필요하며, 이와 같은 취지로 다양한 연구에서 옥트리(octree)를 활용하고 있다(Saxena *et al*., 1995; Woo *et al*., 2002; Wang *et al*., 2004; Schnabel *et al*., 2007; Cho *et al*., 2008; Maréchal, 2009). 특히 포인트 클라우드의 처리를 위한 다양한 기능을 오픈 소스로 제공하고 있는 Point Cloud Library(PCL)에서도 옥트리를 공식적으로 사용하고 있다(PCL, 2015)

한편, 옥트리를 구현하는 방식에 따라 색인 가능한 포인트 클라우드의 용량과 질의 속도에 큰 차이가 발생할 수 있다. 이에 Han(2013)은 C++ 언어로 옥트리를 생성하기 위해 소요되는 메인 메모리의 사용량을 줄이기 위하여 옥트리를 구성하는 노드 클래스(node class)의 변수(member variable)와 자식 노드 접근을 위한 함수(method)의 개선 방안을 제시하였다. 이어 Han(2014a)은 Han(2013)과 달리 포인트 클라우드를 메인 메모리에 저장하지 않고 각 측점이 HDD상에 저장된 위치를 파일 포인터(file pointer)를 통해 참조하는 방법을 사용함으로써 색인 가능한 포인트 클라우드의 용량을 증가시켰다. 다음으로 Han(2014b)은 터널과 같이 종 또는 횡방향으로 현저히 긴 대상물에 대하여 옥트리를 구현하면 질의 효율이 크게 저하되는 문제점을 해결하기 위해 대상물의 3차원 경계를 등축형 하위 영역으로 분할하고 각 영역에 독립적인 옥트리를 생성함으로써 질의 속도를 향상시켰다.

그러나 Han(2014a)과 Han(2014b)의 방식은 각 측점에 대한 파일 포인터를 메인 메모리에 구현된 옥트리의 말단 노드, 즉, 리프 노드(leaf node)에 저장하는 방식으로, 측점 수가 증가함에 따라 메인 메모리의 사용량이 여전히 증가하여 결국 색인 가능한 포인트 클라우드의 용량에 한계가 존재한다. 예를 들어, 64bit 운영체제 환경에서 8byte 크기의 파일 포인터를 사용할 경우 메인 메모리 1GB 당 색인 가능한 측점의 수는 약 1억 3천만 개로 제한되며 옥트리 자체의 용량과 기본적으로 운영체제가 차지하는 용량을 제외하면 실제로 색인 가능한 포인트 클라우드의 용량은 더욱 줄어든다. 아울러 HDD를 기반으로 하는 파일 시스템에서 파일 포인터가 가리키는 임의의 위치에 접근하여 자료를 읽기까지의 시간(random access time)은 메인 메모리에 비해 100배 이상 소요되어 질의 성능 면에서도 상당히 불리하다는 단점이 있다.

따라서 본 연구는 Han(2014b)의 방식을 개선하여 색인 가능한 포인트 클라우드의 용량을 늘리고 질의 성능을 향상시키는 것을 목표로 한다. 이를 위하여 각 리프 노드에 저장되는 파일 포인터의 수를 1개로 제한하고 동일한 리프 노드에 속하는 측점들이 연속적으로 기록되도록 포인트 클라우드 파일을 재구성함으로써, 옥트리 생성에 필요한 메인 메모리의 크기와 측점에 대한 질의 시간을 줄일 수 있는 방식을 제안하였다.

2. 본 론

2.1 기존 연구의 문제점

Han(2014a)과 Han(2014b)의 연구에서는 포인트 클라우드의 측점을 색인하기 위하여 HDD의 파일에 존재하는 각 측점에 대한 파일 포인터를 메인 메모리에 생성된 옥트리의 리프 노드에 저장하는 방식을 사용하였다. Fig. 1에서 offset은 파일의 시작으로부터 측점이 존재하는 곳까지의 차이로서, 8byte 배정밀도 실수(double) 형식의 x, y, z 좌표로 구성된 측점 한 개의 용량은 24byte이므로 offset 값 1은 파일 포인터 값의 차이 24를 의미한다. 32bit 운영체계에서 파일 포인터는 4byte 크기의 변수이므로 관장할 수 있는 파일의 최대 크기는 2^(4×8)byte=4GB이며 이에 해당하는 측점의 수는 178,956,970개이다. 최근 4GB 이상의 고용량 포인트 클라우드 파일이 존재하므로 Han(2014a)과 Han(2014b)의 연구에서는 64bit 운영체계에서 8byte 크기의 파일 포인터를 사용하였으며 이론적으로 2^(8×8)byte=2^24TB의 파일에 존재하는 측점에 대한 접근이 가능하다. 그러나 측점 한 개를 옥트리로 색인할 때마다

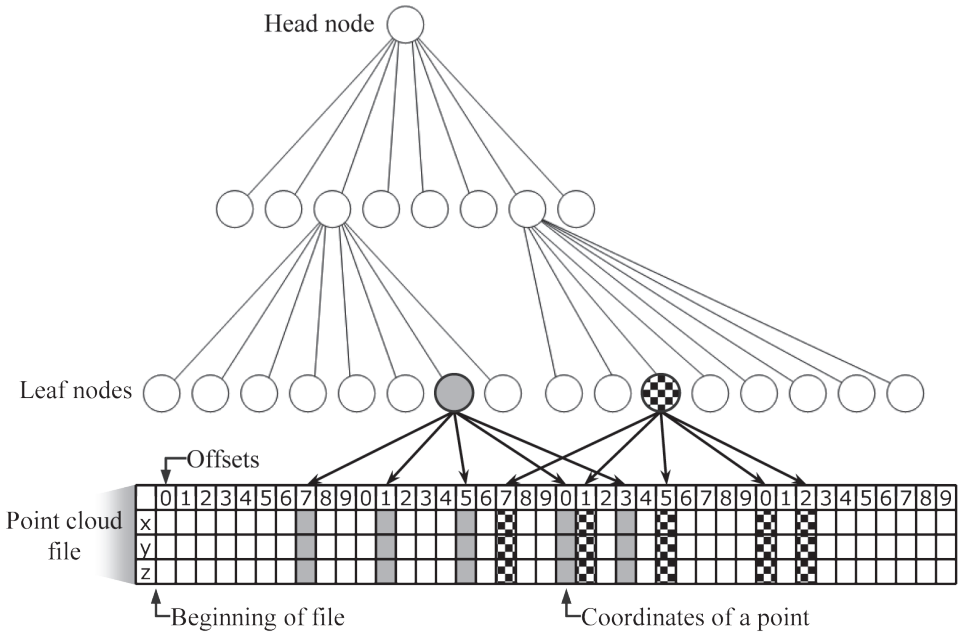

Fig. 1. File-referring scheme introduced in Han(2014a) and Han(2014b)

8byte의 파일 포인터를 리프 노드에 추가함으로써 메인 메모리의 사용량이 증가하고 결국 메인 메모리 1GB 당 색인 가능한 측점의 수는 1GB/8byte=134,217,728개로 제한된다.

다음으로 하나의 리프 노드에 저장되는 측점들은 공간적으로는 인접해 있지만 파일 상에서는 서로 동떨어져 존재할 가능성이 있다(Fig. 1). 실제로 대부분의 3차원 지상 레이저 스캐너는 상하 방향으로 스캐닝을 수행하면서 측점의 좌표를 저장하므로 좌우로 인접한 측점은 파일에서 서로 동떨어진 위치에 저장될 가능성이 있다. 아울러 서로 다른 위치에서 레이저스캐닝을 수행하고 포인트 클라우드를 병합했을 경우 공간적으로는 인접한 측점이 파일 상에서 동떨어지게 저장될 가능성은 더욱 크다. 따라서 동일한 리프 노드에 저장된 측점을 질의할 때에도 HDD 내부에서는 헤드(head)를 서로 동떨어진 위치에 접근시켜야 하는 경우가 빈번히 발생할 수 있다. 이와 같은 헤드의 물리적인 접근 시간과 자료를 읽기 전까지의 지연 시간은 메인 메모리에 비하여 100배 이상 소요되는 것으로 알려져 있다.

2.2 본 연구의 제안

본 연구에서는 포인트 클라우드 파일을 재구성하여 Fig. 2와 같이 동일한 리프 노드에 속하는 측점들을 파일 상의 연속한 위치에 기록하고 리프 노드에는 첫 번째 측점에 대한 파일 포인터만을 저장하는 방식을 제안한다. 이를 위하여 리프 노드의 변수는 Fig. 3과 같은 클래스로 선언하고 '2.2.1 옥트리 생성'과 '2.2.2 파일 재구성'의 두 가지 단계를 수행한다. 파일 재구성이 완료되면 '2.2.3 측점의 질의'와 같이 옥트리로부터 측점을 질의하는 방식을 제안한다.

2.2.1 옥트리 생성

옥트리 생성 단계에서는 포인트 클라우드에 대한 모의 옥트리(pseudo octree)를 생성하면서 각 리프 노드에 저장되는 측점의 수를 기록한다. 즉, 측점이 저장될 리프 노드가 결정될 때마다 해당 리프 노드의 변수 nPoint를 1씩 증가시킨다. 모든 측점에 대하여 상기 과정이 완료되면 재구성될 파일에서 각 리프 노드의 첫 번째 측점이 저장될 위치, 즉, 파일 포인터를 결정한다. 이를 위해 생성된 순서대로 리프 노드를 질의하여 이전까지 질의한 리프 노드의 측점 수를 누적하고 측점 1개의 크기를 곱하여 64bit 파일 포인터 변수 offset_begin에 저장한다. offset_begin의 결정은 Eq. (1)을 따른다.

$$\mathrm{offset\_begin} = \sum_{k=1}^{i-1} \mathrm{nPoint}_k \times \mathrm{sizeof(Point3D)} \qquad (1)$$

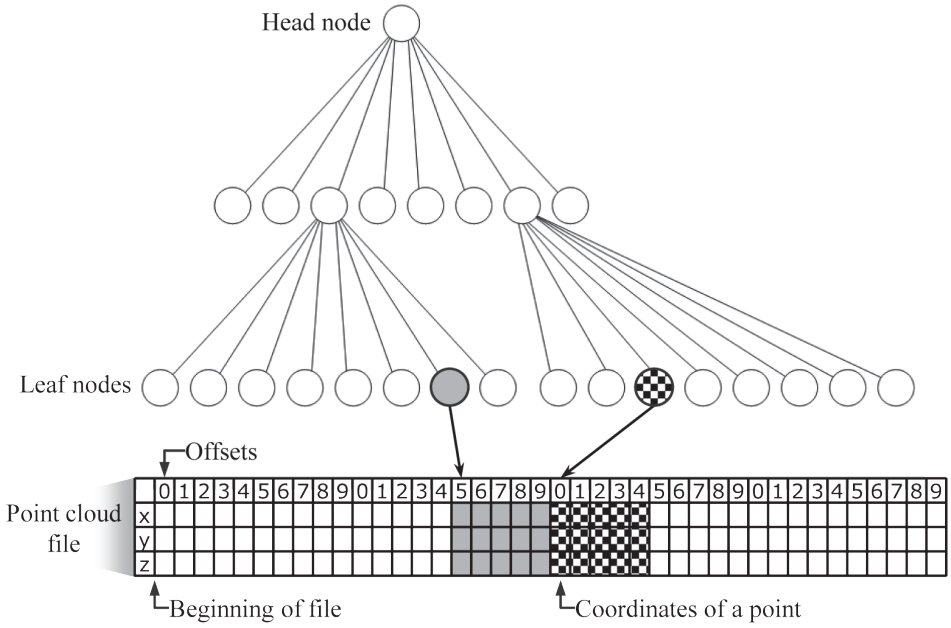

Fig. 2. A new scheme based on re-constructed point cloud file

```
Class CLeafNode{
    int nPoint; // 리프 노드에 속하는 측점의 수, sizeof(nPoint) = 4byte
    int offset; // 첫 측점이 기록된 위치로부터 새로운 측점이 기록될 위치, // sizeof(offset) = 4byte
    __int64 offset_begin; // 파일의 시작으로부터 첫 측점이 기록될 위치, // sizeof(offset_begin) = 8byte
}
```

Fig. 3. Pseudo codes for a new leaf node class

where, i-1 denotes the number of leaf nodes retrieved to reach the current leaf node and sizeof(Point3D) denotes the size of a 3D coordinate point in bytes

2.2.2 파일 재구성

포인트 클라우드 파일을 재구성하기 위하여 기존의 포인트 클라우드 파일과 동일한 용량의 빈 파일(null file)을 생성한다. 다음으로 포인트 클라우드의 각 측점이 속하는 리프 노드를 모의 옥트리로부터 질의하여 offset_begin 값과 offset 값을 받아 Eq. (2)에 따라 최종 위치(offset_final)를 결정하고 재구성된 파일에 측점을 저장한다. 아울러 해당 리프 노드의 offset 값은 다음 측점의 파일 포인터를 결정하기 위해 1씩 증가시킨다. 상기 과정을 모든 측점에 대해 수행하여 파일 재구성을 완료한다.

$$\begin{aligned} offset\_final &= offset\_begin + offset \times sizeof(Point3D) \\ offset &= offset + 1 \end{aligned} \quad (2)$$

where, sizeof(Point3D) denotes the size of a 3D coordinate point in bytes

2.2.3 측점의 질의

주어진 좌표 또는 주위에 존재하는 측점을 질의하기 위해 먼저 주어진 좌표를 포함하는 리프 노드를 질의한다. 해당 리프 노드에 속하는 측점의 수 nPoint에 맞게 배열을 생성하고 첫 측점의 파일 포인터 offset_begin가 가리키는 위치로부터 측점들을 일괄적으로 읽어 배열에 저장한다. 한편, Han(2014a)과 Han(2014b)의 연구에서는 매 측점에 서로 다른 파일 포인터를 이용하여 접근했을 뿐만 아니라 읽은 측점들을 임시 연결 리스트(linked list)에 저장하였다. 따라서 본 연

구에서 제안한 방식은 파일에 저장된 측점들을 읽는 시간과 메모리에 저장된 개별 측점을 질의하기 위한 시간을 함께 감소시키는 효과가 있다.

3. 적용 및 고찰

본 연구에서 제시한 방식을 3차원 지상 레이저 스캐너로 터널에서 취득한 포인트 클라우드에 적용하여 옥트리 생성과 저장 및 복원, 질의 성능 등을 Han(2014b)의 방식과 비교하였다. 포인트 클라우드의 제원은 Table 1과 같으며 처리 시스템의 사양은 Table 2와 같다. 질의 성능을 평가하기 위해 일련의 측점 주위로 일정 반경 안에 존재하는 측점들에 대한 질의 시간을 측정하였다. Han(2014b)의 연구에서는 옥트리를 구성하는 자식 노드의 분화 심도를 나타내는 목표 단계(level)와 노드 리프의 형상 비율(t_i: 가로, 세로, 높이 중 최대, 최소의 비율)에 다양한 값을 적용했지만, 본 연구에서는 Han(2014b)의 연구에서 비교적 우수한 질의 성능과 메모리 사용량을 나타낸 목표 단계 10과 형상 비율 3으로만 실험을 진행하였다.

Table 1. Specifications of point cloud

Terrestrial laser scanner	C10, Leica Geosystems
Scanned object	A train tunnel of 1.5km length
Number of points	300,525,406
File size	6,879 MB
File structure	x, y, z (3 doubles in binary format)

Table 2. Specifications of system

CPU	AMD FX 8150 (8 cores, 3.6GHz)
RAM	8 GB
OS	Windows server 2008 (64bit)
Compiler	C++ 64bit release, Visual Studio 2010
HDD	Avg. seek time(ms) : 8.5/9.5(r/w) Avg. latency(ms) : 4.16

전체 포인트 클라우드의 0.01%에 해당하는 30,053개의 측점으로부터 반경 0.05m 내부에 존재하는 측점을 질의하였으며 Han(2014b)과 본 연구에서 제안한 방식이 동일하게 17,554,511개의 측점을 찾아내었다. 옥트리 생성과 파일 재구성 시간, 옥트리 저장 및 복원 시간, 질의 시간과 메인 메모리 사용량을 측정하였으며 그 결과는 Table 3과 같다. 파일 재구성 과정은 본 연구에서 제안한 방식이므로 Han(2014b)에는 존재하지 않는다.

옥트리의 생성, 저장과 복원, 질의 및 메모리 사용량 등 모든 면에서 제안한 방식이 향상된 성능을 나타내었다. 옥트리의 생성 시간은 기존 방식이 1,760.16초, 제안한 방식이 1,161.44초로 34%가 단축되었고, 옥트리 저장 시간과 복원 시간은 각각 88%와 84%가 단축되었다. 기존 방식은 모든 측점에 대한 파일 포인터를 옥트리의 리프 노드에 저장하므로 옥트리 생성 시간이 길고 옥트리 자체의 크기가 커서 저장 및 복원 시간이 많이 소요되었으나 제안한 방식은 리프 노드 당 1개의 파일 포인터만을 저장하므로 상기와 같은 결과가 나타났다고 볼 수 있다. 질의 시간은 기존 방식이 546.58초, 제안한 방식이 254.58초로 53%가 단축되어 제안한 방식의 질의 속도가 2배 이상 향상되었음을 확인할 수 있다. 이는 공간적으로 인접한 측점을 근거리에 재배치함으로써 HDD 내부에서 발생하는 원거리 접근이 상당 부분 억제된 결과라고 볼 수 있다. 반면, 제안한 방식은 파일 재구성 시간이 4,726.13초가 소요되어 옥트리 구성 시간 등 다른 과정에 비해 많은 시간이 소요되었음을 확인할 수 있다. 그러나 파일 재구성은 1회만 수행하는 과정으로서, 옥트리의 복원과 질의를 반복한다고 가정할 경우 제안한 방식이 유리하다고 볼 수 있다. 또한 질의 시간은 전체 포인트 클라우드의 0.01%인 30,053개의 측점을 기준으로 측정한 것으로 그의 100배인 1%의 측점을 기준으로 질의를 수행한다면 Han(2014b)과 제안한 방식이 대략 54,700초와 25,500초를 소요할 것으로 예상할 수 있으며 그 차이는 29,200초로서 파일 재구성 시간을 충분히 극복할 수

Table 3. Time and main memory usage

Methods	Octree generation (sec)	File reconstruction (sec)	Octree saving (sec)	Octree restoration (sec)	Query (sec)	Main memory (MB)
Han(2014b)	1,760.16	n/a	227.92	84.51	546.58	4,409
Proposed	1,161.44	4,726.13	27.02	13.37	254.58	1,025

있음을 알 수 있다.

메인 메모리의 사용량은 기존 방식이 4,409MB, 제안한 방식이 1,025MB로 77%가 감소되어 메모리 효율성도 4배 이상 증가하였음을 확인할 수 있다. 추가적인 실험은 진행하지 않았지만, 제안한 방식은 포인트 클라우드의 용량이 증가하더라도 리프 노드에 저장되는 파일 포인터가 1개로 유지되므로 목표 단계와 리프 노드의 형상 비율을 유지한다면 메인 메모리의 사용량은 유사하게 유지될 것으로 판단된다. 따라서 메인 메모리의 크기를 크게 상회하는 초대용량 포인트 클라우드로부터 옥트리를 구성하고 측점을 질의하는 것이 가능할 것으로 판단된다.

4. 결 론

본 연구에서는 3차원 지상 레이저 스캐너로부터 취득한 대용량 포인트 클라우드로부터 옥트리를 생성하고 측점을 질의하기 위한 선행 연구의 파일 참조 옥트리 방식을 개선하였다. 이를 위하여 메인 메모리에 구현된 옥트리의 리프 노드에는 첫 번째 측점의 파일 포인터만을 저장하고, 동일한 리프 노드에 속하는 측점들이 연속적으로 기록되도록 포인트 클라우드 파일을 재구성하였다. 결과적으로 옥트리의 생성 시간, 저장과 복원 시간, 질의 시간 및 메모리 사용량 등 모든 면에서 제안한 방식이 향상된 성능을 나타내었다. 특히 질의 속도는 2배 이상, 메모리 효율성은 4배 이상 증가하였다. 따라서 본 연구는 선행 연구의 방식을 명백히 향상시켰다고 판단할 수 있다. 아울러 메인 메모리의 크기를 크게 상회하는 초대용량 포인트 클라우드로부터 옥트리를 구성하고 측점을 질의하는 것이 가능할 것으로 판단된다. 다만, 포인트 클라우드 파일의 재구성에 상대적으로 많은 시간을 소요되어 개선할 여지가 있으며 향후 연구에서는 전반적인 성능 향상을 위해 병렬처리를 도입하고자 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)

Maréchal, 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. (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. (2014a), 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)

Han, S. (2014b), Enhancing query efficiency for huge 3D point clouds based on isometric spatial partitioning and independent octree generation, *Journal of the Korean Society of Surveying, Geodesy, Photogrammetry and Cartography*, Vol. 32, No. 5, pp. 481-486. (in Korean with English abstract)

PCL (2015), Module octree, *Point Cloud Library*, http://docs.pointclouds.org/1.7.1/group__octree.html (last date accessed: 24 October 2015).

RIEGL (2015a), RIEGL VZ-2000 Datasheet, *RIEGL Laser Measurement Systems GmbH*, http://www.riegl.com (last date accessed: 11 August 2015).

RIEGL (2015b), RIEGL VUX-1 Series Infosheet, *RIEGL Laser Measurement Systems GmbH*, http://www.riegl.com (last date accessed: 11 August 2015).

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.

Wikipedia (2015), Brute-force search, *Wikimedia Foundation, Inc.*, https://en.wikipedia.org/wiki/Brute-force_search (last date accessed: 11 August 2015).

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.