Journal of the Korean Society of Surveying, Geodesy, Photogrammetry and Cartography
Vol. 32, No. 2, 109-115, 2014
http://dx.doi.org/10.7848/ksgpc.2014.32.2.109

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

대용량 3차원 포인트 클라우드를 위한 파일참조 옥트리의 구현

Implementation of File-referring Octree for Huge 3D Point Clouds

한수희[1)]
Han, Soohee

Abstract

The aim of the study is to present a method to build an octree and to query from it for huge 3D point clouds of which volumes correspond or surpass the main memory, based on the memory-efficient octree developed by Han(2013). To the end, the method directly refers to 3D point cloud stored in a file on a hard disk drive instead of referring to that duplicated in the main memory. In addition, the method can save time to rebuild octree by storing and restoring it from a file. The memory-referring method and the present file-referring one are analyzed using a dataset composed of 18 million points surveyed in a tunnel. In results, the memory-referring method enormously exceeded the speed of the file-referring one when generating octree and querying points. Meanwhile, it is remarkable that a still bigger dataset composed of over 300 million points could be queried by the file-referring method, which would not be possible by the memory-referring one, though an optimal octree destination level could not be reached. Furthermore, the octree rebuilding method proved itself to be very efficient by diminishing the restoration time to about 3% of the generation time.

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

초 록

본 연구에서는 Han(2013)이 제안한 메모리 효율적인 옥트리를 기반으로 메인 메모리의 크기에 근접하거나 초과하는 3차원 포인트 클라우드로부터 옥트리를 생성하고 3차원 포인트를 탐색하기 위한 방법론을 제시하고자 한다. 이를 위하여 3차원 포인트 클라우드를 메인 메모리에 저장하여 참조하는 방법 대신 하드디스크의 파일을 직접적으로 참조하는 방법을 제안하였다. 아울러 메인 메모리에 구현된 옥트리를 파일로 저장하고 복원함으로써 옥트리 재현 시간을 줄이는 방법을 제안하였다. 메모리참조 방식과 제안된 파일참조 방식을 실제 터널에서 취득한 1800만 개의 3차원 포인트로 구성된 자료와 3억 개로 구성된 자료에 적용하였다. 결과로 옥트리 생성 및 3차원 포인트 탐색 시 1800만 개로 구성된 자료에 대해서는 메모리참조 방식이 파일참조 방식보다 월등히 빠른 속도를 나타내었다. 3억 개로 구성된 자료에 대해서는 메모리참조 방식으로는 옥트리를 생성할 수 없는 반면 파일참조 방식으로는 옥트리 생성 및 3차원 포인트 탐색이 가능하였다. 최적의 탐색 속도를 위한 목표 단계의 옥트리는 생성할 수 없었지만 3억 개가 넘는 3차원 포인트를 탐색할 수 있다는데 의미를 둘 수 있다. 아울러 옥트리를 재현하기 위해 소요되는 시간은 옥트리를 생성하기 위한 시간의 3% 내외로서 제안된 방식이 매우 효율적임을 확인할 수 있었다

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

Received 2014. 02. 12, Revised 2014. 03. 05, Accepted 2014. 03. 08
1) Member, Dept. of Geoinformatics Engineering, Kyungil University (E-mail:scivile@kiu.ac.kr)

1. 서 론

3차원 지상 레이저 스캐너로부터 취득한 3차원 포인트 클라우드(point cloud)는 정밀 계측, 3차원 가상현실 구현, 역설계(reverse engineering) 분야에서 널리 활용되고 있다. 스캐닝 장비의 발달과 저장 장치의 가격 하락에 힘입어 용량이 급격히 증가하고 있는 가운데 3차원 포인트 클라우드를 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 *et al*., 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). 이에 Han(2013)은 옥트리 구현 시 메모리를 더욱 효율적으로 사용하기 위하여 메모리 사용량에 영향을 미칠 수 있는 노드 클래스의 구성 변수와 옥트리 생성 방식을 분석하고 개선 방안을 제시하였다. Han(2013)의 연구에서는 3차원 포인트 클라우드 전체를 메인 메모리에 저장하고 참조함으로써 옥트리 생성 및 포인트 탐색 속도가 매우 빠르지만 많은 메모리가 필요하다는 단점이 있다. 따라서 메인 메모리의 크기에 근접하거나 초과하는 3차원 포인트 클라우드에 대해서는 옥트리를 생성하기가 어렵다. 이의 대안적인 연구로서 Schön *et al*.(2013)은 옥트리를 생성하고 3차원 포인트 클라우드가 저장되어 있는 오라클 공간 데이터베이스(Oracle Spatial DBMS)와 연결함으로써 대용량 3차원 포인트 클라우드에 대한 탐색을 수행하였다.

본 연구에서는 메인 메모리의 크기에 근접하거나 초과하는 3차원 포인트 클라우드로부터 옥트리를 생성하기 위하여 데이터베이스를 사용하거나 3차원 포인트 클라우드를 메인 메모리에 저장하지 않고 하드디스크의 파일을 직접적으로 참조하는 방법을 제안하고자 한다. 아울러 메인 메모리에 구현된 옥트리를 파일로 저장하고 차후에 복원함으로써 옥트리 재현 시간을 줄이는 방법도 제안하고자 한다.

본 논문의 2.1절에서는, 메모리 효율이 향상된 옥트리의 구조와 성능에 대해 Han(2013)의 연구를 요약적으로 설명한다. 2.2절에서는 Han(2013)에서 사용된 메모리를 참조하는 방식과 본 연구에서 제안하는 파일을 참조하는 방식을 비교 설명하며 2.3절에서는 메모리에 구현된 옥트리의 저장과 복원에 대하여 설명한다. 3절과 4절에서는 실제 터널에서 취득한 3차원 포인트 클라우드에 두 가지 방식을 적용한 결과와 결론을 제공한다.

2. 본 론

2.1 메모리 효율적인 옥트리의 구현

옥트리는 계층적 트리 구조의 일종으로 상위 계층인 부모 노드(parent node)가 여덟 개의 자식 노드(child node)와 연결된 구조이며 쿼드트리(quadtree)의 3차원 확장이다. 공간적으로는 Fig. 1과 같이 상위 노드가 관할하는 3차원 공간을 가로, 세로, 높이 방향으로 각각 양분하여 총 8개의 동일한 크기의 공간을 자식 노드가 관할한다. 일반적으로 3차원 포인트 클라우드를 색인화하기 위하여 C++ 언어로 옥트리를 구현할 경우 노드 클래스는 자식 노드에 대한 포인터, 현 노드의 최소 경계 입방체(minimum bounding hexahedral, 이 후 MBH) 정보와 현 노드의 단계(level) 정보를 저장하기 위한 변수들, 실제 포인트를 저장하거나 참조하기 위한 아카이브(archive) 등 다양한 변수를 갖는다. 이 경우 MBH를 구현하기 위한 변수의 형태에 따라 노드 한 개의 크기는 36 bytes(float형 변수의 경우) 또는 60 bytes(double형 변수의 경우)가 되며, 자식 노드의 단계가 증가할 때마다 한 개의 부모 노드에 대하여 자식 노드가 8개씩 생성되므로 메모리 사용량이 크게 증가하는 문제점이 있다(Han, 2013). 이에 Han(2013)은 Fig. 2와 같이 노드에 정의된 MBH 변수와 현 노드의 단계 변수 등을 제거하여 노드의 크기를 8 bytes로 감소시켰으며, 제거된 변수들의 정보를 보전하기 위하여 옥트리 생성 함수인 AddPoint()를 통해 MBH 변수인 mbh와 단계 변수인 level을 부모 노드로부터 전달 받아 갱신 후 자식 노드에게 전달하는 방식을 제안하였다.

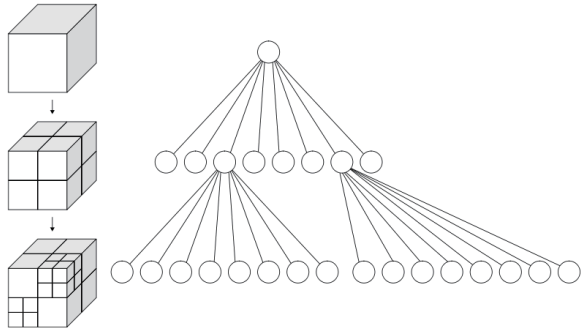

Fig. 1. Octree(left: recursive subdivision of a cube into octants, right: the corresponding octree)(Wikipedia, 2010)

아울러 자식 노드를 생성하거나 리프 노드에서 vector 구조체(저장된 3차원 포인트를 참조하기 위한 아카이브)를 생성할 때, new 연산자 사용을 통한 빈번한 소규모 메모리 동적

```
Struct Point3D{
    double x, y, z }

Class CNode{
    CNode * pChild // 자식 노드에 대한 포인터, sizeof(pChild) = 4 byte
    vector * ptList // vector 구조체에 대한 포인터, sizeof(ptList) = 4 byte

    AddPoint(Point3D *pt, MBH *mbh, int level){
        int n = SelectChildUsing(mbh, pt)
        if (level == DestLevel-1)
            if (pChild[n].ptList == NULL) // 리프 노드에 vector 구조체가 선언되어 있는지 판별
                pChild[n].ptList = new vector // vector 구조체를 동적으로 할당
            pChild[n].ptList.Add(pt) // vector 구조체에 실제 포인트의 포인터 저장
        else
            if (pChild == NULL) // 자식 노드가 선언되어 있는지 판별
                pChild = new CNode[8] // 8개의 자식 노드를 동적으로 할당
            mbh = CalcMBHUsing(mbh, i)
            pChild[n].AddPoint(pt, mbh, level+1)
                // 선택된 자식 노드에 포인트, 갱신된 mbh와 단계 값을 재귀적으로 입력 }
```

Fig. 2. Pseudo codes for MBH-free octree node(Han, 2013)

할당으로 인해 메모리가 추가적으로 소요됨을 확인하였다. 이에 의사 트리(pseudo tree) 생성을 통해 노드 수를 산출하여 모든 노드를 배열로 일괄 선언하고 리프 노드의 vector 구조체 역시 배열로 일괄 선언한 후 정식 트리를 구성함으로써 메모리 효율성을 제고하였다. 메모리 효율적인 옥트리의 구현에 대한 보다 자세한 내용은 Han(2013)을 참조하도록 한다.

2.2 파일참조 옥트리의 구현

Han(2013)의 연구에서는 Fig. 3의 ReadPointCloud() 함수처럼, 하드디스크에 파일로 저장된 3차원 포인트 클라우드 전체를 한 번에 읽어 Point3D 구조체 형의 배열 pPointArray에 입력한다. 옥트리 생성 시 개별 3차원 포인트에 대한 포인터 &pPointArray[i]는 MakeOctree() 함수를 통해 Fig. 2에서 정의한 AddPoint() 함수에 전달된다. 전달된 포인터는 AddPoint() 함수를 통하여 자식 노드에 순차적으로 전달되고 최종적으로 리프 노드의 아카이브인 vector 형식의 ptList에 저장된다. 이러한 방식은 하드디스크에 저장된 3차원 포인트 클라우드 전체를 한 번에 읽어 메인 메모리에 저장하고 옥트리 생성과 포인트 탐색 시 메인 메모리에 저장된 3차원 포인트 클라우드를 참조함으로써 옥트리 생성 및 포인트 탐색 속도가 매우 빠르다. 그러나 옥트리뿐만 아니라 포인트 클라우드 전체를 메인 메모리에 저장함으로써 많은 메모리가 필요하다는 단점이 있다. 결국 메인 메모리의 크기에 근접하거나 초과하는 3차원 포인트 클라우드에 대해서는 옥트리를 생성하기가 어렵다.

```
void ReadPointCloud(){
    Point3D * pPointArray = new Point3D[nPoint]
        // 3차원 포인트의 수(nPoint)만큼 Point3D 형의 배열(pPointArray)로 선언
    fread(pPointCloud, sizeof(Point3D), nPoint, pFile)
        // 포인트 클라우드 파일(pFile)로부터 3차원 포인트를 모두 읽어 배열에 저장 }

void MakeOctree(){
    CNode head;
    for (i=0; i<nPoint; i++)
        head.AddPoint(&pPointArray[i], mbh, level); }
```

Fig. 3. Pseudo codes for making octree

```
void MakeOctree(){
    CNode head;
    for (i=0; i<nPoint; i++)
        Point3D pt;
        int pos = ftell(pFile);
        fread(&pt, sizeof(Point3D), 1, pFile);
        head.AddPoint(pt, mbh, level, pos); }
```

Fig. 4. Pseudo codes for making octree using file pointers

본 연구에서는 메인 메모리의 크기에 근접하거나 초과하는 3차원 포인트 클라우드로부터 옥트리를 생성하기 위하여 옥트리는 메인 메모리에 구현하지만 3차원 포인트 클라우드는 메인 메모리에 저장하지 않고 하드디스크의 파일을 직접적으로 참조하는 방법을 제안하고자 한다. 이를 위하여 Fig. 4와 같이 fread() 함수를 통해 파일로부터 3차원 포인트를 하나씩 읽어 Point3D 형 지역 변수 pt에 일시적으로 저장하되, 그 전에 C standard library의 ftell() 함수를 이용하여 해당 포인트의 파일상 위치, 즉 파일 포인터를 pos 변수에 입력한다. Fig. 2에 정의된 AddPoint() 함수는 3차원 포인트에 대한 포인터 &pt뿐만 아니라 파일 포인터 pos를 함께 전달하도록 수정하고 리프 노드에서는 &pt 대신 pos를 저장하도록 한다. 따라서 트리를 구성하는 노드는 메인 메모리에 형성되지만 포인트 클라우드 자체는 메인 메모리에 존재하지 않으며 각 리프 노드에는 파일의 3차원 포인트를 직접 참조할 수 있는 파일 포인터가 저장된다. 결국 메인 메모리의 크기에 근접하거나 초과하는 3차원 포인트 클라우드에 대해서 옥트리를 생성할 수 있는 가능성이 높아진다. 그러나 옥트리 생성 및 포인트 탐색 시 파일로부터 3차원 포인트를 하나씩 읽음으로써 발생하는 빈번한 파일 I/O로 인해 옥트리 생성 및 탐색 속도가 크게 저하되는 단점이 있다.

2.3 옥트리의 저장과 복원

작업 또는 시스템의 종료에 의해 메인 메모리에 구현된 옥트리가 소멸된 후 다시 구현하기 위해서는 상기 과정들을 반복해야 하나 3차원 포인트 클라우드를 구성하는 3차원 포인트의 좌표가 변경되지 않는 한 옥트리는 동일한 형태로 구현된다. 따라서 메인 메모리에 구현된 옥트리를 하드디스크에 저장하고 필요시 메모리에 복원함으로써 옥트리 재현 시간을 단축할 수 있다. 특히 2.2절에서 제시한 파일참조 옥트리의 경우 옥트리 생성 시 많은 시간이 소요되므로 이와 같은 방법을 통하여 단축시킬 필요가 있다.

옥트리의 저장은 다음과 같이 수행한다. 2.1절에서 설명한 바와 같이 Han(2013)의 메모리 효율적인 옥트리를 구성하는 모든 노드는 배열로 선언되어 메인 메모리에 연속적으로 저장되어 있고 리프 노드의 vector 구조체 역시 동일한 상황이므로 이 두 가지 배열을 하드디스크의 파일로 저장한다. 이 때 각 노드에 존재하는 자식 노드에 대한 포인터와 리프 노드에 존재하는 vector 구조체에 대한 포인터도 함께 저장된다. 그러나 vector 구조체에 저장된 3차원 포인트에 대한 포인터들은 자동으로 저장되지 않으므로 별도의 파일에 각 vector 구조체에 저장된 포인터의 개수와 포인터들을 순차적으로 저장한다.

옥트리의 재현은 다음과 같이 수행한다. 먼저 메인 메모리에 노드 배열과 vector 구조체 배열을 선언하고 파일로부터 일괄적으로 입력한 후 각 vector 구조체에 저장될 3차원 포인트에 대한 포인터들을 별도의 파일로부터 순차적으로 입력한다. 이 때 각 노드에 입력된 자식 노드에 대한 포인터와 리프 노드에 입력된 vector 구조체에 대한 포인터는 옥트리 저장 시 메인 메모리상의 절대 주소를 가리킨다. 그러나 복원 시 노드 및 vector 구조체 배열은 이전과 동일한 위치에 선언되는 것은 아니므로 자식 노드 및 vector 구조체에 대한 포인터는 유효하지 않을 가능성이 높다. 따라서 옥트리 복원 시 배열의 시작 주소와 저장 시 배열의 시작 주소 간 차이(offset)를 이용하여 자식 노드 및 vector 구조체에 대한 포인터를 보정해 주어야 한다.

3. 적용 및 평가

본 연구에서는 Han(2013)과 마찬가지로 3차원 포인트 클라우드로부터 옥트리를 생성하기 위하여 소요된 메모리의 크기와 시간, 3차원 포인트로부터 일정 반경 안에 존재하는 인근 3차원 포인트들에 대한 탐색 시간을 측정하였다. 비교 대상은 3차원 포인트 클라우드 전체를 메모리에 저장하여 참조하는 방식(이후 메모리참조 방식 또는 memory-referring method)과 파일을 참조하는 방식(이후 파일참조 방식 또는

file-referring method)다. 메인 메모리의 용량에 근접하는 대용량 3차원 포인트 클라우드에 대해서는 파일참조 방식만을 적용하였으며 옥트리의 저장 및 재현을 수행하여 각각의 소요 시간을 측정하였다.

약 1800만 개의 3차원 포인트로 구성된 3차원 포인트 클라우드(이후 자료 1 또는 data 1)와 약 3억 개로 구성된 대용량 3차원 포인트 클라우드(이후 자료 2 또는 data 2)에 대한 제원은 Table 1과 같으며 실험에 사용된 시스템의 사양은 Table 2와 같다. 자료 2의 경우, 시스템의 메인 메모리 크기인 8GB보다 작은 6.7GB이지만 실험 당시 시스템의 백그라운드 서비스(background service)에 의해 사용되는 메모리가 1.5GB이어서 메모리참조 방식에는 적용할 수 없었다.

Table 1. Specifications of data

	Data 1	Data 2
Terrestrial laser scanner	SS2, Leica Geosystems	C10, Leica Geosystems
Scanned object	Tunnel	Tunnel
Number of points	18,376,726	300,525,406
File size in MB(GB)	420.6(0.4)	6878.5(6.7)
File structure	3 double variables per a point in binary format	3 double variables per a point in binary format

Table 2. Specifications of system

CPU	AMD FX 8150 (8 cores running at 3.6GHz)
RAM	8 GB
OS	Windows 7 Professional 64bit
Compiler	C++ 64bit release in Visual Studio 2010

먼저 자료 1로부터 옥트리 구성을 위해 소요된 메인 메모리의 크기와 시간, 인근 3차원 포인트 탐색 시간은 Table 3과 같다. 36,754개(전체의 0.2%)의 3차원 포인트로부터 반경 5cm 내에 존재하는 모든 3차원 포인트를 탐색하였으며 탐색 속도가 빠른 수 가지의 옥트리 목표 단계(destination level)에 따른 결과를 분석하였다. 여기서, 옥트리 목표 단계란 옥트리 생성 시 공간 분할 횟수를 의미하며 한 개의 리프노드가 가질 수 있는 최대 부모 수와 일치한다. 목표 단계가 증가함에 따라 리프 노드의 크기가 줄어 탐색 영역이 축소하는 순효과와 탐색 경로가 길어지는 역효과의 조합에 의해 탐색 시간이 감소하다 다시 증가하는 현상을 보인다(Han, 2013). 주어진 반경 안에 탐색된 3차원 포인트의 총 수는 모든 단계에서 15,761,676개로 일관된 결과를 나타내었다.

메모리참조 방식의 경우 기본적으로 3차원 포인트 클라우드의 크기인 420MB를 차지하였으며 목표 단계가 증가하면서 옥트리가 차지하는 메인 메모리의 크기도 증가하였다. 파일참조 방식의 경우 동일한 옥트리 목표 단계에서 메모리참조 방식에 비하여 3차원 포인트 클라우드의 크기만큼 적은 용량의 메인 메모리를 차지함을 확인할 수 있었다. 그러나 옥트리 생성 및 탐색 시간은 파일참조 방식이 월등히 많은 시간을 기록하였다. 최소 탐색 시간은 메모리참조 방식은 목표 단계 9에서 기록한 반면 파일참조 방식은 목표 단계 11에서 기록하였으며 약 45배의 차이를 나타내었다. 두 방식에서 최소 탐색 시간을 기록한 목표 단계의 차이는 리프 노드에서 3차원 포인트에 접근하기 위한 시간의 차이로 해석할 수 있다. 즉, 메모리참조 방식은 리프 노드에서 3차원 포인트에 대한 접근 시간과 부모 자식 노드간 접근 시간이 유사하나 파일참조 방식은 리프 노드에서 파일 I/O에 의한 3차원 포인트에 대한 접근 시간이 부모 자식 노드간 접근 시간보다 월등히 크다. 그러므로 옥트리 목표 단계 증가에 의한 탐색 영역 축소 효과가 메모리참조 방식보다 지속된다.

Table 3. Memory and time occupancy of memory-referring and file-referring methods using data 1

Destination level		Memory-referring		File-referring	
		Building	Querying	Building	Querying
7	Memory(MB)	420+176		Not conducted	
	Time(sec)	4.06	8.74		
8	Memory(MB)	420+186		Not conducted	
	Time(sec)	5.41	3.01		
9	Memory(MB)	420+222		225	
	Time(sec)	6.05	1.75	26.93	216.41
10	Memory(MB)	420+361		361	
	Time(sec)	7.85	2.03	29.87	107.52
11	Memory(MB)	420+825		825	
	Time(sec)	11.29	6.16	32.24	79.62
12	Memory(MB)	Not conducted		1891	
	Time(sec)			38.30	82.85
13	Memory(MB)	Not conducted		3601	
	Time(sec)			45.83	264.44

다음으로 자료 2로부터 옥트리 구성을 위해 소요된 메인 메모리의 크기와 시간, 인근 3차원 포인트 탐색 시간, 옥트리 저장 및 복원 시간은 Table 4와 같다. 30,053개(전체의 0.01%)의 3차원 포인트로부터 반경 5cm 내에 존재하는 모든

Table 4. Memory and time occupancy of memory-referring and file-referring methods using data 2

Destination level		Memory-referring	File-referring			
			Building	Querying	Saving	Rebuilding
9	Memory(MB)	N/A	2744			
	Time(sec)	N/A	1248.15	9975.08	38.64	39.09
10	Memory(MB)	N/A	2913			
	Time(sec)	N/A	1292.55	4536.10	43.12	38.08
11	Memory(MB)	N/A	3364			
	Time(sec)	N/A	1319.25	2573.78	66.22	39.34
12	Memory(MB)	N/A	4668			
	Time(sec)	N/A	1462.40	1563.44	130.40	58.50

3차원 포인트를 탐색하였으며 메인 메모리가 허용하는 한도 내에서 수 가지의 옥트리 목표 단계에 따른 결과를 분석하였다. 주어진 반경 안에 탐색된 3차원 포인트의 총 수는 모든 단계에서 17,554,511개로 일관된 결과를 나타내었다.

인근 3차원 포인트에 대한 평균 탐색 시간은 목표 단계 12에서 89.06㎲/point(= 1563.44sec/17,554,511points)로서, 자료 1에 대하여 목표 단계 11에서 기록한 시간인 5.05㎲/point(= 79.62sec/15,761,676points)과 약 18배의 차이를 나타내었다. 이는 자료 2의 크기가 매우 큰 반면 메인 메모리의 제약에 의해 최적 목표 단계의 옥트리를 생성하지 못한 것으로 해석할 수 있다. 다만, 메모리기반 방식으로는 옥트리를 구현할 수 없는 대용량 3차원 포인트 클라우드로부터 옥트리를 구현하여 3차원 포인트에 대한 탐색을 수행했다는 것에 의미를 둘 수 있다.

옥트리 생성 시간은 목표 단계 9에서 1248.15초를 기록하였으나 저장 시간은 38.64초, 재현 시간은 39.09초를 기록하였으며 다른 목표 단계에서도 재현 시간은 생성 시간의 3% 내외의 시간을 기록하였다. 결국 옥트리의 저장 및 복원 방식이 매우 효율적임을 판단할 수 있다.

4. 결 론

본 연구에서는 Han(2013)의 연구를 기반으로 메인 메모리의 크기에 근접하거나 초과하는 3차원 포인트 클라우드로부터 옥트리를 생성하기 위하여 3차원 포인트 클라우드는 메인 메모리에 저장하지 않고 하드디스크의 파일을 직접적으로 참조하는 방법을 제안하였다. 아울러 메인 메모리에 구현된 옥트리를 파일로 저장하고 차후에 메모리 복원함으로써 옥트리 재현 시간을 줄이는 방법을 제안하였다.

메모리참조 방식과 제안된 파일참조 방식을 터널에서 취득한 1800만 개의 3차원 포인트로 구성된 자료와 3억 개로 구성된 자료에 적용하였다. 결과로 옥트리 생성 및 3차원 포인트 탐색 시 1800만 개로 구성된 자료에 대해서는 메모리참조 방식이 파일참조 방식보다 월등히 빠른 속도를 나타내었다. 3억 개로 구성된 자료에 대해서는 메모리참조 방식으로는 옥트리를 생성할 수 없는 반면 파일참조 방식으로는 옥트리 생성 및 3차원 포인트 탐색이 가능하였다. 최적의 탐색 속도를 위한 목표 단계의 옥트리는 생성할 수 없었지만 3억 개가 넘는 3차원 포인트를 탐색할 수 있다는데 의미를 둘 수 있다. 아울러 옥트리를 복원하는데 소요된 시간은 옥트리를 생성하는 시간의 3% 내외로서 제안된 방식이 매우 효율적임을 확인할 수 있었다.

향후 연구에서는 옥트리 리프 노드의 크기와 저장되는 3차원 포인트에 대한 포인터의 개체수를 조절함으로서 파일참조 방식의 탐색 속도를 제고하고자 한다. 궁극적으로는 3차원 포인트 클라우드의 용량에 영향을 받지 않는 파일기반 옥트리를 구현하고자 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)

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.

Schön, B., Mosa, A. S. M., L. Laefer, D. F., and Bertolotto, M. (2013), Octree-based indexing for 3D point clouds within an Oracle SpatialDBMS, *Computers & Geosciences*, Vol. 51, pp. 430-438.

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 (2010), Schematic drawing of an octree, a data structure of computer science, *Wikimedia Foundation, Inc*., http://en.wikipedia.org/wiki/Octree (last date accessed: 12 February 2014).

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.