Space-Filling Curve Based Point Clouds Index

Jun Wang and Jie Shan

School of Civil Engineering, Purdue University
550 Stadium Mall Drive, West Lafayette, IN 47907-2051, USA
{wang31, jshan}@purdue.edu

Abstract

Managing large volume points clouds data generated from laser scanner is a challenging problem in Geographic Information System (GIS) and spatial database. Based on analyzing the pros and cons of the existing management methods, this paper presents a method to manage lidar data in databases based on the Hilbert space-filling curve. Each lidar data point (X, Y, and Z) is encoded (indexed) by the 3-D Hilbert curve. Data points are organized together according to their Hilbert codes. The initial encoding level of Hilbert curve is determined by the total number of points and the target record size. The data points are first encoded with this initial level Hilbert curve. After refining and combining processes, the data volume of each group is controlled under the desired size. One record in database represents one data group; the binary blob of the record contains all the data points in one group. Details on constructing 3-D Hilbert curve are discussed. Typical query process "window query" is implemented. Reported in this paper are results based on synthetic and real lidar data collected from ground tripod lidar and airborne lidar equipments.

1. Introduction

LIDAR (Light Detection and Ranging), including both airborne and ground-based laser scanning, is currently a widely used remote sensing technology for fast acquisition of precise and reliable 3-D spatial information. Point clouds generated from laser scanner have been used in many different geo-information areas, such as digital terrain model generation, 3-D modeling of urban environment and landscape analysis (Ackermann, 1999; Palmer and Shan, 2002).

Point clouds data can be represented as multi-dimensional arrays such as 3-D (Cartesian coordinates: X, Y, Z) and 4-D (Cartesian coordinates and Intensity of returned pulse). Although existing CAD and GIS software can directly import these arrays into main memory as point features, handling millions of data points usually exceeds their computing capacity. The points cloud dataset requires a significant storage capacity, and the loading time of the dataset from files or databases can be unbearable. It is crucial to provide advanced managing and query functions in lidar data handling such that the interest subset of data can be rapidly located and read from the secondary storage. Present database management (DBMS) systems do not provide a simple way to manipulate multi-dimensional arrays; the operations on arrays are very limited and not optimized (Marathe

and Salem, 2002). Efficient organization, storage and retrieval of lidar data have posed a challenging problem for many practical applications.

A common practice to manage the lidar data is to partition the space the lidar data resides into regular tiles (e.g., 1 mile by 1 mile) or grids (such as orthophoto grid or township grid) and then store the lidar data in one tile/grid as one single file in ASCII or binary format. The large volume of dataset is divided into separated files with a reasonable size. The grid itself can be stored as shapefile and the links to the external lidar data files are stored in the corresponding attribute table (Merrick, 2004; NCFMP, 2004). However, it is very difficult to perform efficient queries and retrievals. For example, if one wants to access the data points with their elevations in certain range, all the data in the query area has to be downloaded from data sever then imported to CAD/GIS software to conduct the query to find the desired data points. For large working areas, the loading time can take up to tens of minutes. As an alternative, the lidar data can be directly stored in databases; each data point is inserted into database as a single record. Since the data is stored and accessed in single point level, it is easy to perform the query and analysis on the data sever, and only the interest data points are returned to users. This method works well for small dataset up to several millions of points. The downside of this method is that high investment on software and hardware is needed to effectively manage a large database for a lidar project.

To overcome the shortcomings of the above two methods, we propose to apply the space-filling curves to partition the lidar dataset. The partitioned dataset will be stored in a spatially indexed relational database. First, the space in which lidar data is embedded is partitioned into a number of 3-D dimensional cells. The extension of the 3-D cells varies according to the lidar data density, such that the number of lidar points in each cell is as even as possible or closes to a predefined target value. Next, the 3-D cells are ordered in space based on the principle of Hilbert space-filling curve. In this way, cells with adjacent sequential numbers (Hilbert codes) will also be adjacent in space. The lidar points within the same 3-D cell are then stored as a binary blob (Binary Large Object), the data (blob and Hilbert code) of 3-D cells are input into a database table in the order of the Hilbert codes. For spatial query, a hierarchical strategy is applied that initially determines the large cells where lidar points may possibly reside and continues to reach smaller cells until all qualified data points are found. Under this management strategy, the balance between storage requirements and fast-query needs is achieved.

The rest of this paper is organized as follows. Firstly, the related studies are reviewed in Section 2. Section 3 introduces Hilbert space-filling curve and its representation based on permutation rules. Main topic of this paper, indexing and querying lidar data with Hilbert curve, is presented in section 4. Section 5 presents the results from current implementation, followed by conclusion remarks in Section 6. Details on constructing 3-D Hilbert curve are listed as Appendix.

2. Related Works

Spatial data, including lidar data, are usually organized as 2-D, 3-D or even higher dimensional arrays. To store the high dimensional data in 1-D media, such as harddisk,

the mapping between higher dimensional space and 1-D space has to be developed. For spatial data, there is no perfect mapping such that any two spatially adjacent objects are also adjacent to each other in 1-D storage media (Gaede and Gunther, 1998). Many hierarchical spatial data structures (also called multidimensional access methods) have been developed for organizing and representing spatial data in GIS and spatial databases. All these methods fall in two categories: point access methods and spatial access methods. Point access methods only organize spatial points. It first partitions the space into different areas according to certain criteria, and then groups the points by areas. This category is also called space-partition based access methods. Spatial access methods are spatial object (such as points, lines and polygons) based approach. Most methods in this category use MBB (minimum bounding box) of the objects, and are thus referred as rectangle access methods (Gaede and Gunther, 1998). There is no single optimal spatial data structure that is suitable for all kinds of spatial data. Each method has its strengths and limitations. In GIS and spatial databases related applications, Quadtree (Octree in 3-D case) (Samet, 1989) and R tree (also several variations, such as R^+ tree, R* tree) (R: Guttman, 1984; R^+: Sellis, et al., 1987; R*: Beckmann, et al., 1990) are two most widely used methods. The performance of these two methods has been compared in Oracle database using GIS data (Kothuri, et al., 2002): for point data, Quadtrees are faster in index creation/update and has the storage requirements nearly the same as R-trees; R-trees are slower than Quadtrees for certain types of spatial queries on point layers. Some preliminary studies (Brinkhoff, 2004; Wang and Tseng, 2004) show that spatial access methods have the potential use for organizing lidar data and performing feature extraction.

For lidar dataset, no matter which spatial data structure is applied, the dataset has to be divided into different groups by certain space-partition mechanism. An Octree-like space-partition method based on 3-D Hilbert space filling curve is proposed in this paper.

3. Hilbert Space-Filling Curve

Space-filling curves (Sagan, 1994) map points in N-dimensional space into a 1-D linear order. The curve visits each point in space only one time in a certain order - usually points that are close on the curve are close in space. There is no perfect mapping to preserve global spatial proximity. Space-filling curves preserve spatial proximity at local level to some extent; the closer two object in space, the higher possibility that they are close together in the linear order defined by space-filling curves. Because of the characteristics of mapping between one and N-dimensions and the distance-preserving (or clustering) property, the space-filling curves are especially useful in applications that involve the storage and retrieval of multi-dimensional data with 1-D media (e.g., hard disk) (Gaede and Gunther, 1998).

A space-filling curve can be used with a space partition method. A high-dimensional space can be divided into different grid cells, which can be in turn further divided into smaller cells until the cell size or the number of interest objects in the cell is small enough. The level of such partition depends on the smallest cell size and the number of grid nodes in space that space-filling curve can pass through. Each cell is labeled by the unique number (called code) that defines cell’s position in the order of space-filling curve. The way of labeling determines the order in which the cells are stored in 1-D media.

There are many different types of space-filling curves, such as Hilbert, Peano (N-ordering curve), Gray, Sweep, C-Scan and Diagonal etc (Mokbel, et al., 2003). The 2-D Hilbert curve and Peano curve are shown in Figure 1. To evaluate the clustering abilities of space-filling curves, one simple instinct way is to identify the jump segments (two consecutive points are not in the von Neumann neighborhood of each other); the lack of jump segments is usually a good indication for better clustering. We can see that jump segments exist in Peano curve. Peano curve separates the data space into more small pieces (clusters) than Hilbert curve. Both mathematical analysis and practical applications suggest that Hilbert curve has best clustering ability and performance in data retrieval and response time among all kinds of space-filling curves. (Faloutsos and Roseman, 1989; Lawder, 1999; Moon, et al., 2001; Mokbel, et al., 2003). However, due to the simplicity of its mapping functions, Peano curve is also widely used in spatial data management applications (Pascucci and Frank, 2001).

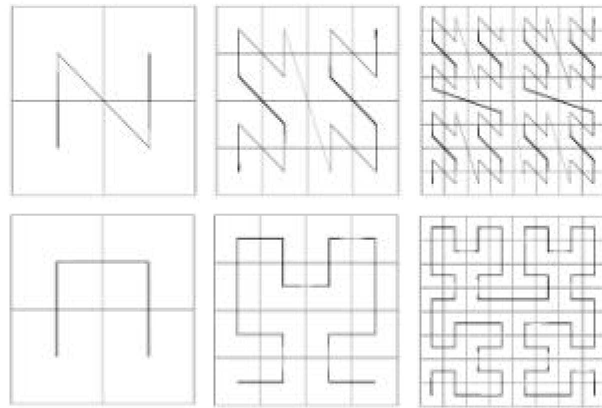

Figure 1. 2-D Peano (top) and Hilbert (bottom) curves

Based on the above observations, 3-D Hilbert curve is used to order the partition cells and manage the lidar data (X, Y, Z or Northing, Easting, Elevation) in this study. The examples of 3-D Hilbert curves from level 1 to level 3 are shown in Figure 2.

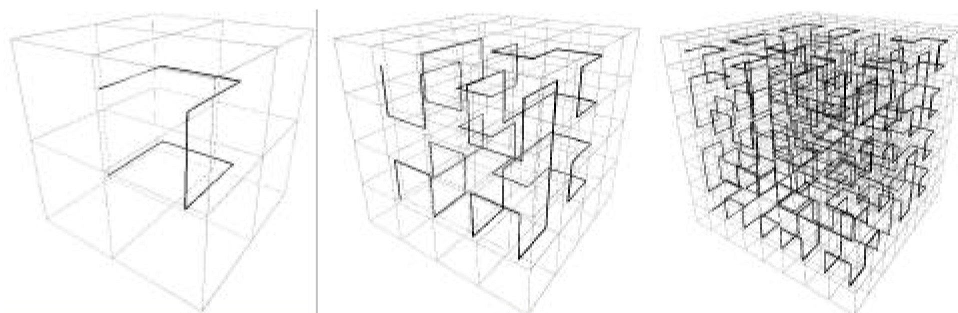

Figure 2. 3-D Hilbert space-filling curve (Level 1, 2, 3 from left to right)

3.1 Generation of Hilbert curves

Due to the self-similarity property of the Hilbert curves, higher level Hilbert curve can be generated recursively from the lower level curve. The generation can be done with a set of recursive rules. Many researchers have studied recursive generation of Hilbert curve in 2-D and 3-D space, different methods, such as recursion (Breinholt and Schierz, 1998), vertex labeling (Bartholdi and Goldsman, 2001), L systems (Alfonseca and Ortega, 1996), tensor product (Lin, et al., 2003) and table driven method (Jin and Mellor-Crummey, 2005) have been developed. For space higher than three dimensions, it is more difficult to

find rules to generate Hilbert curves. Non-recursive methods, usually based on byte-oriented operations, have been developed for generating Hilbert curve in higher dimensional space (Butz, 1971; Lawder, 2000). Although technically they can be used to generate Hilbert curves in any dimensions, these non-recursive methods are too complicated and not as much efficient as recursive methods in lower dimensional space. There is only one unique form of Hilbert curve in 2-D space, however, there are many different ways to define Hilbert curve in three or higher dimensional space (Sagan, 1993). According to Alber and Niedermeier (2000), there are exactly 1536 structurally different CHPs (Class with Hilbert Property) in 3-D space.

Since lidar data can be represented in 3-D or 4-D arrays, the recursive generation methods is used in this research. After carefully studying all these different methods, we summarize that recursive generation procedures can be expressed in the following way: a d-dimension cube is divided equally into 2^d sub-cubes; the first level Hilbert curve ($H^d{}_1$) is generated by connecting the center of each sub-cube in the vertex labeling order. The set of permutation rules will partition a $H^d{}_{k-1}$ level cube into 2^d $H^d{}_k$ cubes. So a d-dimension Hilbert curve can be recorded as the ordered 2^d vertices and 2^d permutation rules. The permutation rule takes the following basic form:

$$\begin{pmatrix} V'_1 \\ V'_2 \\ \dots \\ V'_{2^d} \end{pmatrix} = \frac{1}{2}\begin{pmatrix} a_{11} & a_{12} & \dots & a_{12^d} \\ a_{21} & a_{22} & \dots & a_{22^d} \\ \dots & \dots & \dots & \dots \\ a_{2^d1} & a_{2^d2} & \dots & a_{2^d2^d} \end{pmatrix}\begin{pmatrix} V_1 \\ V_2 \\ \dots \\ V_{2^d} \end{pmatrix} + \frac{1}{2}\begin{pmatrix} b_1 \\ b_2 \\ \dots \\ b_{2^d} \end{pmatrix} \quad (1)$$

where ***[V]*** and ***[V']*** are the ordered vertices of a d-dimension cube and its sub-cube, respectively, ***[a]*** is the rotation/reflection matrix, ***[b]*** is the shifting vector, and the scale factor is 0.5. The rules can be written down in 2-D by direct observing the geometry formation of Hilbert curve. We first apply this method to 2-D Hilbert curve, and then provide one example rule for 3-D Hilbert curve. All 8 permutations rules in 3-D case are listed in Appendix. Figure 3 shows the situation of $H^2{}_1$ (the first level curve in 2-D).

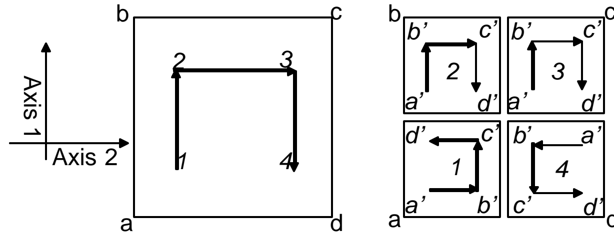

Figure 3. Recursive generation rules for 2-D Hilbert curve

For sub-cell 2 and 3, vertex 2 and 3 are connection points for constructing $H^2{}_1$ from two H^1_1, the edge 2→3 are parallel to the axis 2, the rotation/reflection matrix for both sub-cell are identity matrix. The permutation rules for these two sub-cells are:

$$\text{subcell 2: } \begin{pmatrix} a' \\ b' \\ c' \\ d' \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} + \frac{1}{2} \begin{pmatrix} b \\ b \\ b \\ b \end{pmatrix} \qquad (2)$$

$$\text{subcell 3: } \begin{pmatrix} a' \\ b' \\ c' \\ d' \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} + \frac{1}{2} \begin{pmatrix} c \\ c \\ c \\ c \end{pmatrix} \qquad (3)$$

For the sub-cell 1, it is generated by the reflection along the vertex a and its diagonal vertex c, these two vertices will keep their positions, and the other two vertex b and d on the diagonal line which is perpendicular to a→c will exchange their positions. The permutation rule for the cell is:

$$\text{subcell 1: } \begin{pmatrix} a' \\ b' \\ c' \\ d' \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} + \frac{1}{2} \begin{pmatrix} a \\ a \\ a \\ a \end{pmatrix} \qquad (4)$$

In a similar manner, the permutation rule for sub-cell 4 is:

$$\text{subcell 4: } \begin{pmatrix} a' \\ b' \\ c' \\ d' \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} + \frac{1}{2} \begin{pmatrix} d \\ d \\ d \\ d \end{pmatrix} \qquad (5)$$

In this way we obtain the permutation set for 2-D Hilbert curve.

In 3-D, the cube will be divided into 8 sub-cell recursively and there are 8 permutation rules for 3-D Hilbert curve. Figure 4 shows the example of $H^3{}_1$ and the mapping of the first sub-cell (which includes vertex a). The rule for first sub-cell is listed below (see Appendix for all 8 rules).

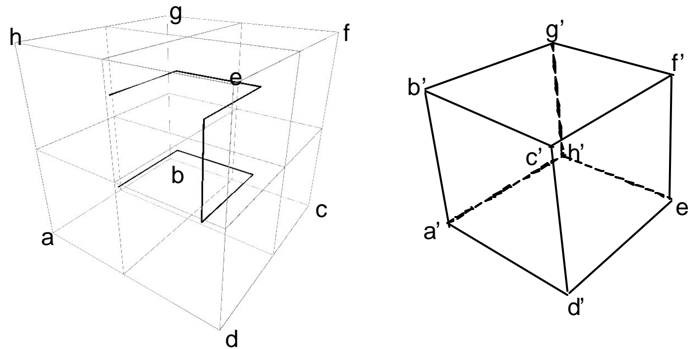

Figure 4. First level 3-D Hilbert curve and mapping of the first sub-cell

$$\text{subcell 1:}\begin{pmatrix} a' \\ b' \\ c' \\ d' \\ e' \\ f' \\ g' \\ h' \end{pmatrix} = \frac{1}{2}\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}\begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{pmatrix} + \frac{1}{2}\begin{pmatrix} a \\ a \\ a \\ a \\ a \\ a \\ a \\ a \end{pmatrix} \quad (6)$$

3.2 Encoding and Decoding

The 3-D Hilbert curve maps 3-D points into a 1-D linear order. The sequential number (position) of a point on the curve is called Hilbert code. Each Hilbert code has the corresponding enclosing cell in 3-D space. Data points are ordered according to the sequence in which the curve visits the cells that enclose the data points. Each cell can be assigned to base-4 digit ([0-3]) in 2-D and base-8 digit ([0~7]) in 3-D to represent its position relative to the parent (next lower level) cell. The 3-D Hilbert code is a string of base-8 digits, the length of string equals to the coding level.

The operations of converting between Hilbert code and 3-D coordinates are referred as encoding and decoding. Figure 5 shows an example of encoding a point to level 3 in 3-D space.

Encoding procedure

1. Given a point (x,y,z) and the 0-level cube;
2. Find the closest vertex i among all 8 vertices of the cube;
3. Set i as Hilbert code for this level;
4. Apply ith permutation rule to get 8 vertices of the sub-cell which contains the point;
5. Repeat step 2-4 until desired coding level;
6. Return Hilbert code string.

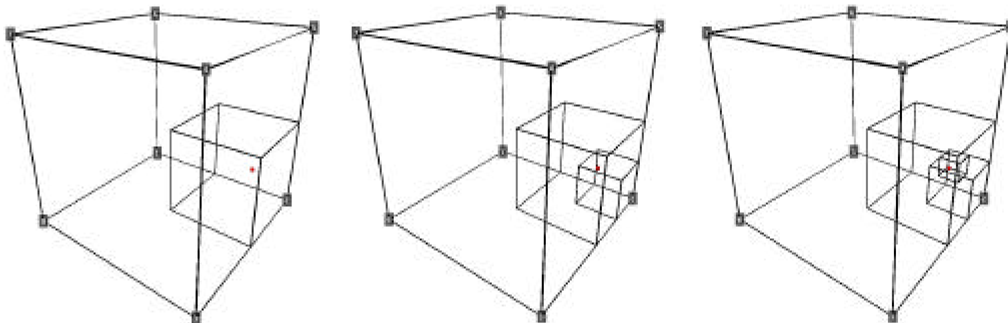

Figure 5. The process of encoding a point to level 3
Level 1 code: “2”, level 2 code: “2”, level 3 code: “6”, the final code is “226”

Through a decoding process, we can find the enclosing cell of the given Hilbert code.

Decoding procedure

1. Get the 0-level cube;
2. $i = 1$;
3. Pick up the ith digit from Hilbert code string;
4. Apply ith permutation rule to get the vertices of the sub-cell;
5. $i = i + 1$;
6. Repeat step 3-5 until $i >$ length (code);
7. Return the vertices of current cell.

4. Indexing and Querying Lidar Data

4.1 Indexing

Indexing procedure can be described as a recursive space partition process. We first divide the space into cells (small 3-D region) with certain level Hilbert curve. If the number of lidar points in a cell is larger than the predefined target value, it will be further divided into 8 higher-level sub (smaller) cells (the next higher level in Hilbert curve). This process is repeated until no cell contains more than the target number of data points. In the next step, the rest of never-be-divided cells are summed up according to the parent cell (the next lower level in Hilbert curve). If the total number of data points in cells with the same parent is less than the target value, then all these cells are combined into one larger cell (parent cell). After these refining and combining processes, the lidar points are input into a database with each cell being stored as one record. The data points in the same cell are stored in the binary blob of the corresponding record.

Indexing procedure

1. Choose initial encoding level;
2. Encode data points with initial level;
3. Count number of data points with the same Hilbert code;
4. Generate the list of Hilbert code for refining process;
5. Generate the list of Hilbert code for combining process;
6. Run refining procedure until list from step 4 is empty;
7. Run combining process until list from step 5 is empty;
8. Group data by Hilbert code;
9. Insert grouped data points into database.

The complexity of the algorithm depends on k and n, k is the largest encoding level and n is the number of data points, since k << n, the real indexing time is in general proportional to the number of data points. The detail of indexing procedure is explained as following five steps: initialization, encoding, refinement, combine and input to database.

In the initialization step, we first choose the target size and decide the initial Hilbert curve encoding level. The target size is the maximum number of points stored in each record for a database. It can be chosen based on hardware performance and Internet speed or the size of most frequently queried study area. A target size of several K to several 10K will

be a proper choice for most applications. The initial Hilbert curve is determined by the following equation

$$NL = \text{int}(\log_8 Pnts / Ts + 1.5) \quad (7)$$

where NL is the initial encoding level, $Pnts$ is the total number of lidar points, Ts is the target size. For example, we have 1 million points, and the target size is 2000. If data points are evenly distributed in 3-D space, there will be 500 records. A level-3 Hilbert curve can pass through 512 (8^3) cells and therefore is sufficient. However, due to the non-uniform distribution of the data points, we choose level 4 as initial coding level, since level 4 can pass through 4096 (8^4) cells. Choosing the proper initial encoding level according to the distribution of points in 3-D space can reduce the time spent on the following refining and combining processes.

In the encoding step, all the data points are encoded into the initial encoding level. It should be noted that it is not necessary to encode each lidar point by exactly following the procedure described in Section 3.2. The lidar sampling and file are sequential and the possibility for one point to be in the same cell with the previous point is high. Each point is therefore compared with its previous point. If they are in the same cell, this point will have the same Hilbert code as the previous point; otherwise, it will go through the complete encoding procedure. During this process, the number of data points in each cell is counted. If this number is larger than the target number, then the Hilbert code of this cell is added to the list of cells that need further division or refinement. If the number of data points in the cell is less than the target number and none of its adjacent cells is in refining list, this cell is added to the candidate list to be combined.

For the cells in the refining list, all the points in a cell are passed into the encoding procedure again and encoded into the next higher level. This is equivalent to dividing the cell further into 8 sub cells in 3-D space. Then the number of points in each sub-cell is counted, if it is higher than the target size, this sub-cell is further divided. Repeat this process until all the cells contain points no more than the target size. If initial encoding level is too low, most cells need to be further divided into several levels deeper in the Hilbert curve. Repeated encoding processes is time consuming for large data set, since exacting data points from large arrays is a slow process.

The combing step is to merge certain cells based on their number of enclosing points. After refining process, the 3-D space partition contains cells in different sizes. For the cells which are never divided in the refining process, their Hilbert code length is equal to the initial encoding level. All these cells will be summed up according to their parent cell. Testing if certain cells belong to the same parent is straightforward, for example, cell '0324' and '0325' are in the same parent cell '032'. If the total number is less than the target size, the cells are combined into the parent cell, and the Hilbert code of the parent cell will be assigned to the points in these cells. If the parent cell contains refined cell, then all the sub cells in this parent cell are not combinable. After scanning all the never-divided cells, the lower level parent cell will be scanned again, until all the cells are not combinable. If initial encoding level is too high, each cell is very small and may contain

lidar points much less than the target size. As a consequence, most cells need to be combined into lower level parent cells. Since there is no encoding process involved, it is less time consuming than the refining process.

Figure 6 shows an example of 3-D space grids after each step from a small set of testing lidar dataset of tree canopy (see Table 1 in Section 5). For illustration purpose, the range of XYZ coordinates have been normalized into (0, 1). From the left to right, Figure 6 shows the grid state after each step. The initial encoding level is 3; all the cells have the same level of codes (Green box represent level 3 cell). After the refining process, some cells are coded into level 4 (blue cell). In the combining step, the undivided cells are counted; some of them are combined to level 2 cells (Yellow cell).

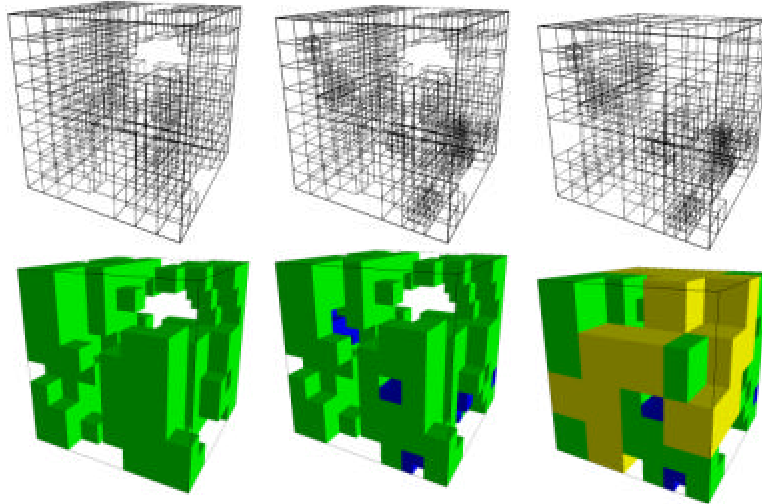

Figure 6. The grid state after each step, 1) encoding step, 2) refining step and 3) combining step; Yellow box: level 2, Green box: level 3, Blue box: level 4.

The final step is to write the encoded data into a database. Any relational database can be used to store lidar data. The basic table structure contains a string column and a binary blob column, which are respectively for the Hilbert code and its associated data points.

4.2 Querying

Among different types of spatial queries, this paper implements the most frequently used window (range) query. It is the process to find all the data points inside a given 3-D cube. We first encode the 8 vertices of the query region until all of them have different Hilbert codes. This means the query region has been divided into 8 cells. Only data cells inside these 8 cells need to be considered. The database is then scanned to retrieve the subset of data cells. If the cell from the subset is inside the query window, all the data points in the cell are selected; if the cell overlaps with the query window, each data point in the cell is scanned to decide if it is inside the query window. Since the data points in each cell are no more than the target size, it is not a time consuming process to perform this scanning process.

5. Evaluation

Both indexing and querying procedures are evaluated. The lidar data is stored into MySQL database and Microsoft Access. Python is used as the development language. Both synthetic and real lidar data are used in our study. Synthetic lidar data is created as 3-D random points with uniform distribution, while the three sets of real lidar data are respectively collected by airborne lidar equipment over the Purdue campus, and by ground tripod lidar for two bridges at the interstate highway I-70 in Indianapolis, Indiana. Table 1 lists the properties of the test data sets and the time needed for spatial indexing and query. The relationship between indexing time and number of data points shows in Figure 7. The tests are performed with a Pentium IV 2 GHz desktop with 512M memory.

Table 1. Test lidar sets, encoding results and query performance

	Synthetic-1	Synthetic-2	Tree	Campus	Bridge-1	Bridge-2
Data source.	Random generated in Matlab	Random generated in Matlab	Tree canopy Scan	Airborne, Purdue Campus	I-70 bridge Optec, north side	I-70 bridge Optec, the whole bridge
Total points	20,000	200,000	4,296	620,738	372,294	1,427,043
Target points	100	2,000	50	3,000	3,000	10,000
Initial level	4	4	3	4	3	4
Refined points	6,422	64,381	1,089	76,451	327,682	326,874
Combined points	2,381	12,332	553	29,041	15,579	74,904
Indexing time	132 s	323 s	10 s	950 s	750 s	2,800s
Window query time*	<1 s	<1 s	<1 s	<3 s	<3 s	<5 s

*Window query time is the average time of 100 random queries

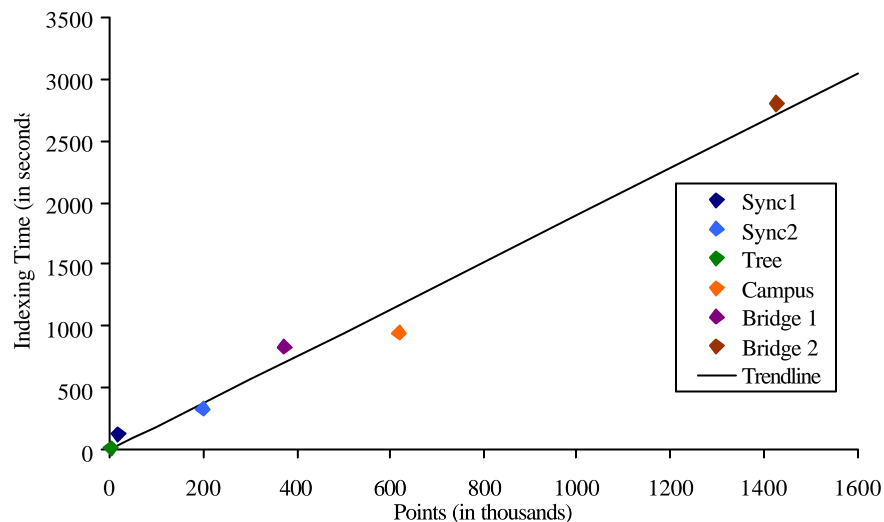

Figure 7. Indexing time vs. number of points

Figure 8 shows the results of Bridge-1 data set. The distribution of data points is extremely non-uniform. After the initial encoding at level 3, 88% of the total points need to be refined, the repeated encoding process in refining step slows down the data processing greatly, and one way to solve this problem is to increase the initial level. How to decide the optimal initial encoding level for the extremely non-uniform data needs to be investigated in future study.

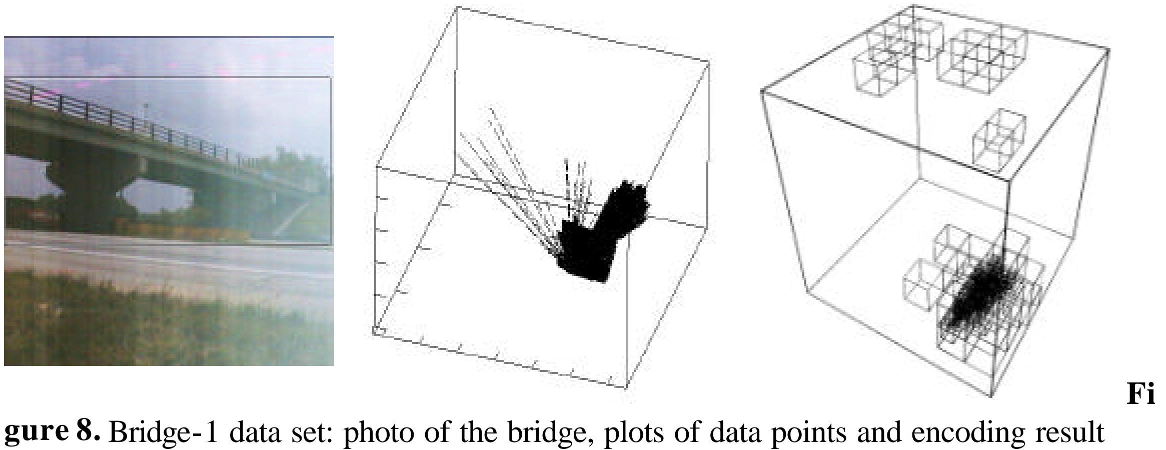

Figure 8. Bridge-1 data set: photo of the bridge, plots of data points and encoding result

The result from campus data set is shown in Figure 9. Compared with Bridge-1 data set, the density distribution of campus data set is much more close to uniform. Its indexing time would be less than non-uniform distributed Bridge-1 data set if both of them had the same number of data points.

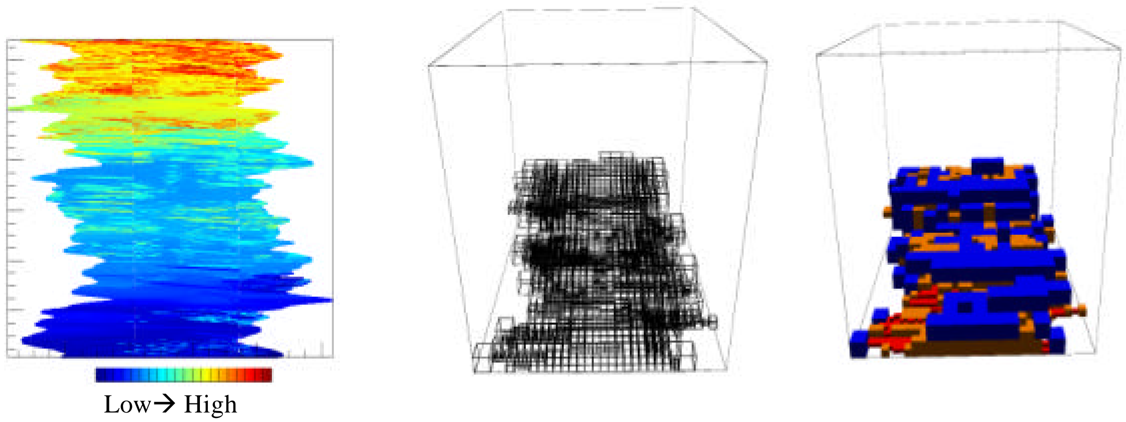

Figure 9. Campus data set: data points on XY plane (color represents elevation), encoding result in wire frame and encoding results in box format (color represents encoding level)

6. Conclusions

The goal of this work is to study the theory and methodology to effectively manage large volume lidar data in spatial database. Octree partition and the Hilbert space-filling curve

are used for this purpose. The ordinary 2-D Hilbert curve is expanded to a 3-D formulation to support true 3-D spatial index and query. It is shown that a 3-D Hilbert curve can be generated through a set of matrices by repeatedly applying rotation and reflection to a basic matrix. To spatially index the lidar data, the 3-D space is recursively partitioned into cells based on octree principle till the number of lidar points in each cell is no more than a predefined target value. Lidar points are then written into a relational database cell by cell as a blob by following its order on the Hilbert curve. In this way, the lidar points are nearly equally distributed among the cells and therefore the record of the spatial database can be kept as the same manageable size. Study with two synthetic and three real lidar datasets validates the developed theory and methodology. It is shown that such indexing is extremely beneficial for 3-D window query. Our experience suggests that the distribution of lidar data has great effect on the indexing performance. Difficult arises in determining the initial partition level and target size for extremely uneven distributed lidar data. Future effort will be focused on further reducing the time needed for indexing calculations by optimizing the encoding algorithm. Moreover, the presented methodology will be expanded to 4-D to support both location and intensity related query.

Appendix. Permutation rules for 3-D Hilbert curve

We adopt the method suggested by Moon, et al. (2001) to construct 3-D Hilbert curve. This method first defines the orientation of Hilbert curve: for any level of Hilbert curve, the coordinates of starting point and ending point only differ in one dimension. The curve is i-orientated if both points lie on a line parallel to i-th coordinate axis. The Hilbert curve can be recorded as ordered 2^d vertexes and 2^d orientations of their corresponding sub cells. Therefore, what needed is to generate permutation rules from orientations. For example, the orientation series for 2-D Hilbert curve (see Figure 3) is 1-2-2-1 for vertex 1 to 4, respectively.

3-D curve can be constructed from two 2-D curves (see Figure A-1). The orientation series of 3-D curve are the same as the ones in 2-D except the two vertices connected by the vertical line parallel to the newly introduced axis 3. Since the orientations for these two vertices change from 1-ortiented to 3-ortiented, the orientation series of this 3-D curve is 1-2-2-3-3-2-2-1 (first 2-D curve: 1-2-2-1, second 2-D curve: 1-2-2-1).

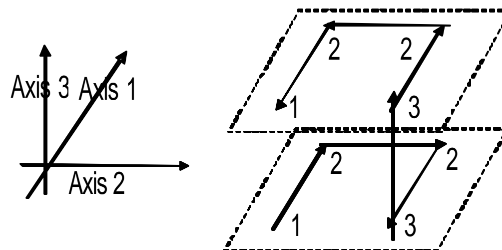

Figure A-1. Construct 3-D curve from two 2-D curves (the label stands for orientation)

Permutation rules are generated from the orientation series (1-2-2-3-3-2-2-1) for each vertex. Since for the i-th rule the sub cell is shifted to i-th vertex, the shifting vector can be omitted in the following expressions, where only reflection/rotation matrix is shown.

For rule 1, the orientation is 1, the reflection/rotation matrix is:

$$\text{subcell 1:} \begin{pmatrix} a' \\ b' \\ c' \\ d' \\ e' \\ f' \\ g' \\ h' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{pmatrix} \quad \text{(A-1)}$$

For rule 2, the orientation 2, the reflection/rotation matrix is:

$$\text{subcell 2:} \begin{pmatrix} a' \\ b' \\ c' \\ d' \\ e' \\ f' \\ g' \\ h' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{pmatrix} \quad \text{(A-2)}$$

For rule 3, the orientation is 2, the reflection/rotation matrix is the same as rule 2.

For rule 4, the orientation is 3, the reflection/rotation matrix is:

$$\text{subcell 4:} \begin{pmatrix} a' \\ b' \\ c' \\ d' \\ e' \\ f' \\ g' \\ h' \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{pmatrix} \quad \text{(A-3)}$$

The 3-D curve is constructed by mirroring the two 2-D curves by the plane perpendicular to the axis 3 (see Figure A-1). The above 4 rules are the mapping of sub cells on the first 2-D curve. The mapping of 4 sub cells on the second 2-D curve mirrors the mapping of these on the first 2-D curve. The rest four rules on the second 2-D curve can be obtained by making a transpose operation along the lower diagonal of the corresponding reflection/rotation matrix of the above 4 rules. For example, rule 7 is obtained from rule 2 by this transpose operation.

$$
\text{subcell 7:} \begin{pmatrix} a' \\ b' \\ c' \\ d' \\ e' \\ f' \\ g' \\ h' \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{pmatrix} \quad \text{(A-4)}
$$

Only three (Equation A-1, A-2 and A-3) out of eight rules are distinct, the other five rules are either the same as these three rules or their transpose. This gives out a simple way to construct Hilbert curve.

References

[1]. Ackermann, F. (1999). Airborne laser scanning - present status and future expectations. ISPRS Journal of Photogrammetry & Remote Sensing 54(1): 64-67.

[2]. Alber, J. and R. Niedermeier (2000). On Multidimensional Curves with Hilbert Property. Theory of Computing Systems 33(4): 295-312.

[3]. Alfonseca, M. and A. Ortega (1996). Representation of fractal curves by means of L systems. Proceedings of the conference on Designing the future, Lancaster, United Kingdom: 13-21.

[4]. Bartholdi, J. J. and P. Goldsman (2001). Vertex-labeling algorithms for the Hilbert spacefilling curve. Software—Practice & Experience 31(5): 395-408.

[5]. Beckmann, N., H.-P. Kriegel, R. Schneider, and B. Seeger. (1990). The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles. Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data, Atlantic City, NJ: 322-331.

[6]. Breinholt, G. and C. Schierz (1998). Algorithm 781: generating Hilbert's space-filling curve by recursion. ACM Transactions on Mathematical Software (TOMS) 24(2): 184-189.

[7]. Brinkhoff, T. (2004). Spatial Access Methods for Organizing Laserscanner Data. XXth International Society for Photogrammetry and Remote Sensing (ISPRS) Congress, Istanbul, Turkey: 98-102.

[8]. Butz, A. R. (1971). Altenative algorithm for Hilbert's space filling curve. IEEE Transactions on Computers 20: 424-426.

[9]. Faloutsos, C. and S. Roseman (1989). Fractals for secondary key retrieval. Proceedings of the eighth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, Philadelphia: 247-252.

[10]. Gaede, V. and O. Gunther (1998). Multidimensional access method. ACM Computing Surveys (CSUR) 30(2): 170-231.

[11]. Guttman, A. (1984). R-Trees: A Dynamic Index Structure for Spatial Searching. SIGMOD'84, Proceedings of Annual Meeting, Boston, Massachusetts: 47-57.

[12]. Jin, G. and J. Mellor-Crummey (2005). SFCGen: An Framework for Efficient Generation of Multi-dimensional Space-filling Curves. ACM Transactions on Mathematical Software 31(1): 120-148.
[13]. Kothuri, R. K., S. Ravada and D. Abugov. (2002). Quadtree and R-tree indexes in oracle spatial: a comparison using GIS data. Proceedings of the 2002 ACM SIGMOD international conference on Management of data, Madison, Wisconsin: 546-557.
[14]. Lawder, J. K. (1999). The Application of Space-Filling Curves to the Storage and Retrieval of Multi-dimensional Data (PhD), Birkbeck College, University of London.
[15]. Lawder, J. K. (2000). Research Report JL1/00: Calculation of Mappings between One and n-dimensional Values Using the Hilbert Space-filling Curve, School of Computer Science and Information Systems, Birkbeck College, University of London.
[16]. Lin, S.-Y., C.-S. Chen, L. Liu and C.-H Huang. (2003). Tensor Product Formulation for Hilbert Space-Filling Curves. 2003 International Conference on Parallel Processing, Kaohsiung, Taiwan
[17]. Marathe, A. P. and K. Salem (2002). Query processing techniques for arrays. The International Journal on Very Large Data Bases 11(1): 68-91.
[18]. Merrick (2004), Merrick Advanced Remote Sensing (MARS®) Software version 3.2, http://www.merrick.com/
[19]. Mokbel, M. F., W. G. Aref and I. Kamel. (2003). Analysis of Multi-Dimensional Space-Filling Curves. GeoInfomatica 7(3): 179-209.
[20]. Moon, B., H. V. Jagadish, C. Faloutsos and J. H. Saltz. (2001). Analysis of the Clustering Properties of the Hilbert Space-Filling Curve. IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING 13(1): 124-141.
[21]. NCFMP (2004), North Carolina Floodplain Mapping Program. http://www.ncfloodmaps.com/
[22]. Palmer, T. C. and J. Shan (2002). A Comparative Study on Urban Visualization Using LIDAR Data in GIS. URISA Journal 14(2): 19-25.
[23]. Pascucci, V. and R. J. Frank (2001). Global static indexing for real-time exploration of very large regular grids. Proceedings of the 2001 ACM/IEEE conference on Supercomputing, Denver, Colorado.
[24]. Sagan, H. (1993). A three-dimensional Hilbert curve. International Journal of Mathematics Education in Science and Technology 24(4): 541-545.
[25]. Sagan, H. (1994). Space-filling Curves. New York, Springer-Verlag.
[26]. Samet, H. (1989). Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS, Addison-Wesley Publishing Company.
[27]. Sellis, T., N. Roussopoulos, and C. Faloutsos. (1987). The R+-tree: A dynamic index for multidimensional objects. Proceedings of 13th International Conference on Very Large Data Bases, Brighton, England: 507-518.
[28]. Wang, M. and Y.-H. Tseng (2004). Lidar Data Segmentation and Classification Based on Octree Structure. XXth International Society for Photogrammetry and Remote Sensing (ISPRS) Congress, Istanbul, Turkey