Technical Disclosure Commons

Defensive Publications Series

June 05, 2019

Hilbert k-D Tree Encoding

Michael Hemmer

Igor Guskov

Follow this and additional works at: https://www.tdcommons.org/dpubs_series

Recommended Citation

Hemmer, Michael and Guskov, Igor, "Hilbert k-D Tree Encoding", Technical Disclosure Commons, (June 05, 2019)
https://www.tdcommons.org/dpubs_series/2247

This work is licensed under a Creative Commons Attribution 4.0 License.
This Article is brought to you for free and open access by Technical Disclosure Commons. It has been accepted for inclusion in Defensive Publications Series by an authorized administrator of Technical Disclosure Commons.

Hilbert k-D Tree Encoding

Detailed Description

A problem in storing point cloud data is as follows: given a large set of points, find a representation that uses as little information as possible. The generation of the representation (encoding) and the interpretation of the representation (decoding) should be fast so that a user is not waiting an excessive amount of time to render the point cloud data. Algorithms used to encode and decode should be as simple as possible.

Some point clouds are with some 2- or 3-dimensional space often include various additional attributes. In some implementations, attributes include color, estimated normal direction, and reflectivity; there are a plethora of attributes that may be considered. Each of these attributes adds potentially several additional dimensions. For example, an RGB color attribute adds three dimensions, if an alpha channel representing a transparency factor is included, the number of dimensions added is four. The addition of such attributes can result in a point cloud that uses an enormous amount of information; the cost of transmitting such information over a network can be very high.

Conventional encoding algorithms use a k-D tree compression algorithm, which works in arbitrary dimensions. Nevertheless, the K-D tree compression algorithm may become less efficient when the point cloud has a high dimension such as when the point cloud is associated with attributes such as the attributes discussed above.

Improved techniques of encoding point clouds include, in conjunction with the above-described k-D tree encoding, an algorithm that sorts points along a space-filling curve such as a Hilbert curve and uses subsequent delta encoding to compress the point cloud. In general, while the k-D tree approach provides better compression rates, the above-described Hilbert curve approach advantageously uses, as the decoder, a simple and very fast delta decoder. Moreover, a Hilbert curve may use only a subset of the dimensions while sorting; remaining dimensions resulting from attributes are simply data included with point.

As an example, one may sort the points according to the attribute of three-dimensional spatial coordinates, while the attribute of color is not used. However, points that are close in space are also more likely to be similar in color. As a consequence, the delta encoding of the color attribute usually becomes better too, even though the point cloud is only sorted with respect to the spatial coordinates.

The above-described Hilbert curve approach advantageously enables one to compress each attribute separately. This implies that one can load the attributes on demand and encode/decode the information concerning the attributes in parallel.

FIGS. 1, 2, and 3 illustrate various Hilbert curves for different levels of subdivision. The diagrams illustrate how the Hilbert curves are used to order points in a plane. Nevertheless, in many implementations, the point clouds are in three or more dimensions; in such cases, three-or-more-dimensional Hilbert curves, analogous to the two-dimensional curves, may be used to sequence the subdivisions and therefore the bit strings that are the coordinates of those subdivisions.

FIG. 1 is a diagram that illustrates an example space-filling curve of order one. In this case, a point cloud has only two points within a bounding box. The bounding box is subdivided in each direction once. A corresponding Hilbert curve 112 for these subdivisions 110(1), 110(2), 110(3), and 110(4) is illustrated. In this case, the Hilbert curve makes an upside-down "U" shape as it traverses exactly once through each subdivision.

One of the points of the point cloud has coordinates (0,0) and lies in the subdivision 110(1), while the other point has coordinates (1,1) and lies in the subdivision 110(3). The Hilbert curve traverses the subdivisions beginning at the subdivision 110(1) and ending at the subdivision 110(4). Accordingly, the sequence of bit strings in this case is (0,0), (1,1).

FIG. 2 is a diagram that illustrates another example space-filling curve of order two. In this case, there is another point in the subdivision 210(3). Accordingly, one may divide this subdivision into four further subdivisions 220(1), 220(2), 220(3), and 220(4). As illustrated in FIG. 2, one point is in subdivision 220(2) with coordinates (10,11) and the other is in subdivision 220(4) with coordinates (11,10).

In the example illustrated in FIG. 2, each of the rest of the other subdivisions 210(1), 210(2), and 210(3) is also divided into four further subdivisions. For example, the point shown in FIG. 1 in subdivision 110(1) is now in subdivision 230(2) with coordinates (01,00). In other implementations, however, only the original subdivision 110(3) would undergo further subdivision.

A Hilbert curve of order two may be generated from the curve of order one as follows. In subdivisions 220(2) and 220(3), the curve 112 is copied into each subdivision as portions of the order two curve in each of those subdivisions. In the subdivision 220(1), the curve 112 is copied into the subdivision and rotated by -90 degrees as the portion of the order two curve in that subdivision. In the subdivision 220(4), the curve 112 is copied into the subdivision and rotated by +90 degrees as the portion of the order two curve in that subdivision. To form a continuous curve, the ends of each of these portions may be joined with beginnings of adjacent portions; these joinings are shown as dashed lines in FIG. 2.

One may identify a portion 222 of the Hilbert curve with a point or subdivision, e.g., subdivision 220(2). This portion is of order two (corresponding to a second subdivision of the bounding box) and is oriented with zero degrees rotation with respect to the order one Hilbert curve. Based on that information, the space-filling curve manager arranges the bit strings in the ordered sequence (01,00), (10,11), (11,10).

FIG. 3 is a diagram that illustrates an example space-filling curve of order three 3. In this case, there is another point in the subdivision 220(2). Accordingly, one may divide this subdivision into four further subdivisions 340(1), 340(2), 340(3), and 340(4). As illustrated in FIG. 3, one point is in subdivision 340(1) with coordinates (000,110) and the other is in subdivision 340(3) with coordinates (001,111).

In the example illustrated in FIG. 3, each of the rest of the other subdivisions 220(1), 220(3), and 220(4) is also divided into four further subdivisions. For example, the point shown in FIG. 2 in subdivision 220(4) is now in subdivision 350(2) with coordinates (010,001). The point in subdivision 230(2) is now in 360(3). The point in 220(4) is now in 370(2). In other implementations, however, only the original subdivision 110(3) would undergo further subdivision.

A Hilbert curve of order three may be generated from the curve of order two as follows. The entire curve of order two is copied into subdivisions 220(2) and 220(3) so that a copy of the curve of order two is a portion of the order three curve in each of those subdivisions. In the subdivision 220(1), the curve of order two is copied into that subdivision and rotated by -90 degrees as the portion of the order three curve in that subdivision. In the subdivision 220(4), the curve of order two is copied into that subdivision and rotated by +90 degrees as the portion of the order three curve in that subdivision. To form a continuous curve, the ends of each of these portions may be joined with beginnings of adjacent portions; these joinings are shown as dashed lines in FIG. 3.

One may identify a portion 342 of the Hilbert curve with a point or subdivision, e.g., subdivision 340(1). This portion is of order three (corresponding to a three subdivision of the bounding box) and is oriented with zero degrees rotation with respect to the order one Hilbert curve. Based on that information, the space-filling curve manager arranges the bit strings in the ordered sequence (010,001), (010,110), (001,111), (101,111), (110,101).

Once the sequence of bit strings has been identified, one may perform a compression operation on the sequence. First, one performs a delta encoding on the sequence to produce a delta sequence. Then one performs an entropy encoding on the delta sequence to produce a compressed delta sequence. When the sequence of bit strings is arranged according to the Hilbert curve, the compressed delta sequence takes up relatively little storage space because the delta sequence typically has very little variation.

The conventional k-D tree approach may be modified such that it produces the order of the Hilbert curve during decoding. This k-D approach in its basic form first involves counting the number of points in the bounding box and then divides the first axis exactly in the middle. Then the k-D approach records the number of points on the left side, thereby revealing also the number of points on the right side. The k-D tree approach repeats this division with each half and about an axis corresponding to another dimension. Accordingly, for example, after performing the above-described operations for each axis of a three-dimensional point cloud the space has been subdivided into 8 cubes.

The crucial observation is now that we do not have to process these cubes in a naive depth-first search (DFS) (or breadth-first search (BFS)) fashion. Instead the divided boxes may be queued for further subdivision according to the order of the Hilbert curve, which then implies that the order in which points are decoded is the order of the Hilbert Curve. Because this corresponds to a simple reordering of the numbers are subsequently entropy encoded this will not have an effect on the resulting compressed file size because the entropy does not change.

Alternatively, in some implementations, one sorts the decoded points (just their key values) along the Hilbert curve after decoding. Using the convention defined by ordering points according to the Hilbert curve, one can use the same idea to provide the value attributes in a separate encoding. The main issue in this case is that the points must be sorted after decoding, which is usually avoided since the decoding process is usually the time critical one. Note that sorting of a huge point cloud can be time critical, specifically since a Hilbert Curve comparison predicate is significantly slower than an ordinary lexicographic one.

The above-described improved techniques have the following advantages:

- Similar compression ratio as the KDtree approach: The key attributes are compressed using the modified KDtree algorithm. Since, it modifies only the order in which the points are encoded and decoded and not the fundamentals of the algorithm, the compression rate for that part is exactly the same good compression rate as the version without axis selection. That is, the size is the one of the compressed point cloud without the value attributes.
- The order along the Hilbert curve has the advantage that points that are close on the Hilbert curve are also close in memory and also likely to be close in space. Specifically, this is a cache friendly ordering which is beneficial for subsequent processing like nearest-neighbors graphs as they are for instance produced by a probabilistic roadmap method (PRM). In other words, if a neighbor is already loaded into the cache it is likely that a neighbor has been already loaded into memory as well, with the result of fewer cache misses. More precisely the order is cache oblivious since the recursive nature of the Hilbert curve provides this benefit without being explicitly designed for a specific cache size.
- The provided canonical ordering by the Hilbert curve allows the other value attributes to be encoded with delta encoding using the order induced by the order of the key attributes. For instance, the colors or the normals of a points that are close in space are likely to be close in space as well. This spatial coherence then benefits the encoding of each value attribute.
- A major advantage is that the improved techniques enable the provision of certain attributes in a separately encoded file or a separate database column. Accordingly, these attributes do not have to be considered contemporaneously and can be provided on demand, e.g., if needed by a specific application, or say, in case a user would like to also fetch and visualize the color of the point clouds at a later point.

In some implementations, one obtains the uncompressed attribute values only of some points from the server since their index is known, for example only indices of those points that have intersections by a ray tracer etc. Again, the cache oblivious nature of the hilbert curve may be useful here.

As mentioned earlier, the improved techniques are not limited to attributes such as colors or normals. For example, when the encoding time is not critical, in some implementations the encoder may try several combinations to find a combination that provides an optimal compression ratio. Accordingly, every dimension (e.g. the three position coordinates) can be treated separately as key or value attribute.

For certain input types, it may be best to leave the order untouched and use delta encoding right away. Specifically, for LiDAR data before Simultaneous Localization and Mapping (SLAM) has taken place, the order of points and the corresponding timestamp is extremely important. Fortunately, this usually implies some coherence in other attributes.

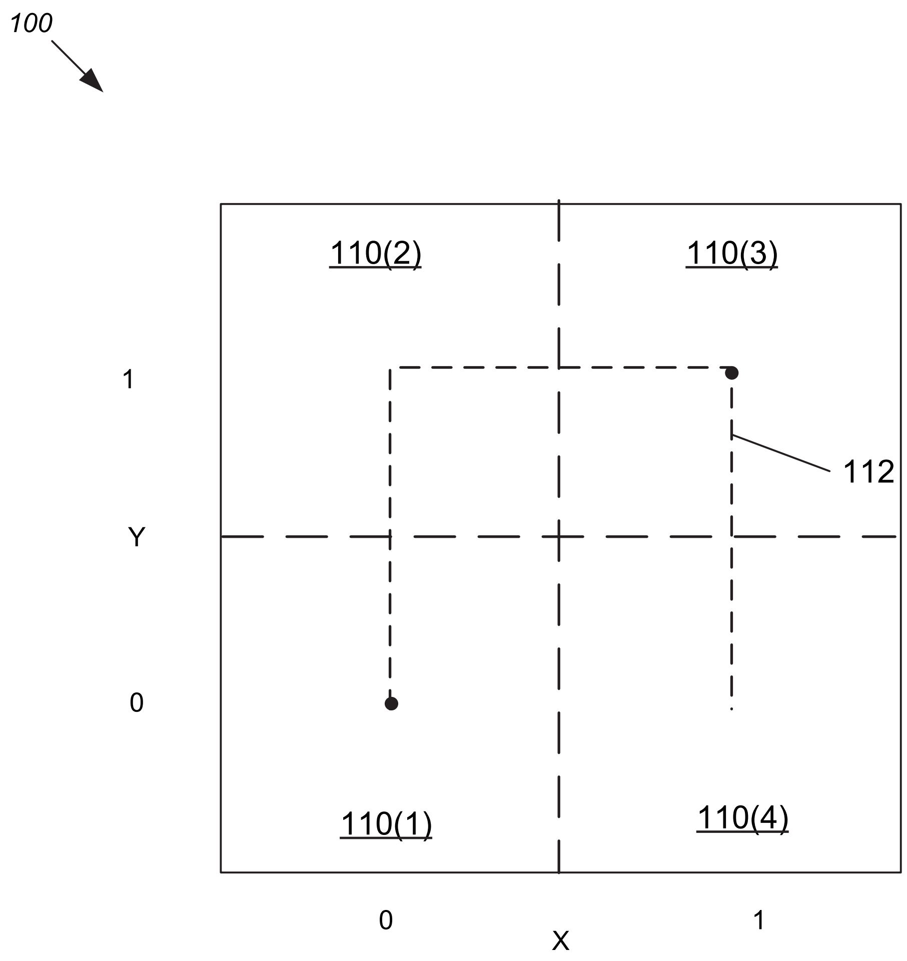

FIG. 1

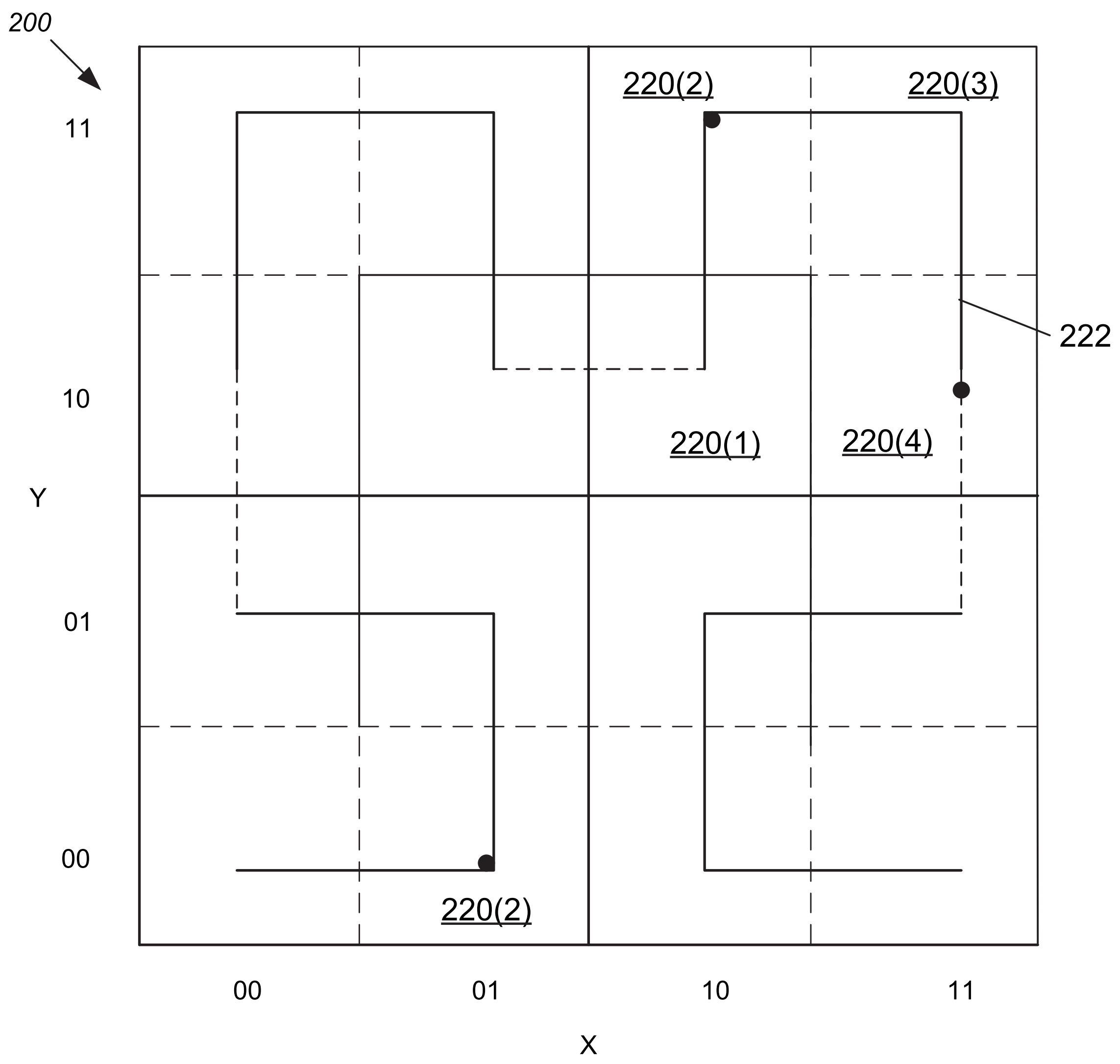

FIG. 2

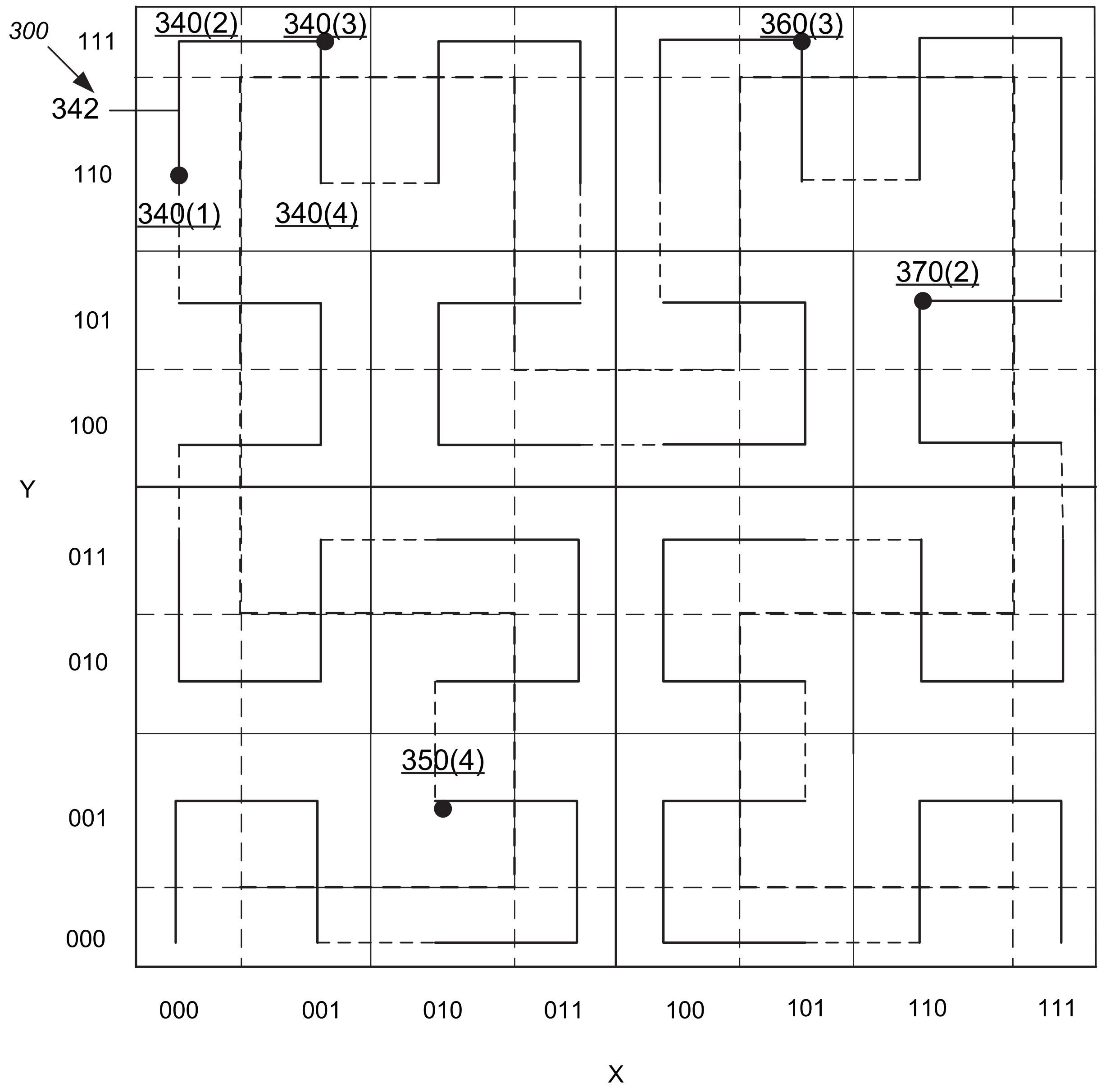
300
342
340(1)
340(2)
340(3)
340(4)
360(3)
370(2)
350(4)
111
110
101
100
011
010
001
000
Y
000
001
010
011
100
101
110
111
X

FIG. 3