Compact Hilbert Indices

Chris Hamilton

Technical Report CS-2006-07

July 24, 2006

Faculty of Computer Science
6050 University Ave., Halifax, Nova Scotia, B3H 1W5, Canada

Compact Hilbert Indices

Chris Hamilton
chamilton@cs.dal.ca

July 5, 2006

Abstract

Space-filling curves are continuous self-similar functions which map compact multi-dimensional sets into one-dimensional ones. Since their invention they have found applications in a wide variety of fields [12, 21]. In the context of scientific computing and database systems, space-filling curves can significantly improve data reuse and request times because of their locality properties [9, 13, 15]. In particular, the Hilbert curve has been shown to be the best choice for these applications [21]. However, in database systems it is often the case that not all dimensions of the data have the same cardinality, leading to an inefficiency in the use of space-filling curves due to their being naturally constrained to spaces where all dimensions are of equal size.

We explore the Hilbert curve, reproducing classical algorithms for their generation and manipulation through an intuitive and rigorous geometric approach. We then extend these basic results to construct *compact Hilbert indices* which are able to capture the ordering properties of the regular Hilbert curve but without the associated inefficiency in representation for spaces with mismatched dimensions.

1 Introduction

Space-filling curves are continuous one-to-one functions which map a compact interval to a multi-dimensional unit hypercube. Originally formulated by Giuseppe Peano in 1890 [24], the first space-filling curve was constructed to demonstrate the somewhat counter-intuitive result that the infinite number of points in a unit interval has the same cardinality as the infinite number of points in any bounded finite-dimensional set. Since their invention, space-filling curves have found applications in a variety of fields, including mathematics [7], image processing [16, 27], image compression [20], bandwidth reduction [23], cryptology [19], algorithms [25], scientific computing [9, 13], parallel computing [2, 14], geographic information systems [1] and database systems [5, 15, 18]. Notably, space-filling curves have also found application in quickly computing approximate solutions to the travelling salesman problem, with this approach leading to the development of a low-complexity delivery vehicle routing system [3].

Since Peano introduced the first space-filling curve numerous others have been constructed and extensively studied. Among these further developments is the family of curves generated by Hilbert [11], which to this day finds many applications. Due to the recursive geometric nature of the original construction, the Hilbert curves naturally impose an ordering on the points in finite square lattices. In particular, the Hilbert curve used in this manner has been found to be the best space-filling curve for preserving data locality [21]. As such, it has been the focus of much research, with numerous algorithms constructed to compute it [4, 6, 7, 8, 12, 17, 22, 26], each directed towards a particular application.

In the first part of this paper we recreate Butz's classic algorithm [8] for Hilbert curves, but from a completely rigorous geometric point of view. This intuitive approach allows for a deeper level of understanding of the primitives used in Butz's algorithm, at the same time providing insight into other algorithmic approaches such as Bartholdi and Goldsman's vertex-labelling approach [4] and Jin and Mellor-Crummey's recent table-driven methods [12].

By considering the order on which the curve visits the points in an n-dimensional lattice with 2^m points per dimension, we may assign an index to each point between 0 and $2^{mn} - 1$. In the context of database systems this enumeration is used to sort the points while preserving *data locality*, meaning that points close in the multi-dimensional space remain close in the linear ordering. This in turn translates to data structures with excellent range query performance [21].

In the real world, not all dimensions are of equal size and consequently the space in which the data points reside may be significantly smaller than the full lattice of side lengths 2^m. As such the Hilbert indices require mn bits to represent, which may be significantly larger than that required to represent the points in their native space. In the second part of this paper we explore the notion of *compact Hilbert indices*, which assign to points an index whose representation requires the same space as that required to represent the points in their native space.

2 Hilbert Curves

The results of this section construct from the ground up the necessary tools for the exploration of Hilbert curves. We take a largely geometric approach, yielding algorithms that are essentially identical to those of Butz in the classic paper [8], whose standard implementation was created by Thomas [26] and later refined by Moore [22].

The terminology and notation used in this section is largely my own, and I have chosen to deviate from existing convention in order to highlight the geometric approach taken here, and the relationship to standard boolean operators. Notation will be introduced upon first use, but for the reader's convenience a complete table of notation may be found in Appendix A. The need for the level of detail in this construction will become apparent in the construction of algorithms for compact Hilbert indices in Section 3.

We consider first the traditional recursive definition of the two dimensional Hilbert curve. For reasons that will become apparent later, we consider the Hilbert curve that starts in the bottom left corner and finishes in the upper left[1]. The curve is initially defined on a 2×2 lattice, as shown in Figure 1. Given the order k curve defined on a $2^k \times 2^k$ lattice, we may refine it to visit all points on a $2^{k+1} \times 2^{k+1}$ lattice as follows:

- Place a copy of the original curve, rotated counter-clockwise by 90°, in the lower left sub-grid.
- Place a copy of the original curve, rotated clockwise by 90°, in the upper left sub-grid.
- Place a copy of the original curve in each of the right sub-grids.
- Connect these four disjoint curves in the obvious manner.

This construction may be visualized in Figure 2, with the first four iterations of the construction shown in Figure 3. In a completely analogous manner one may define the Peano curve, which travels through lattices of size $3^k \times 3^k$ as shown in Figure 4.

Any finite approximation of the Hilbert curves allows a simple mapping from 2-dimensions into 1, by simply associating a given lattice point with its index along the curve. This same concept can be extended to arbitrary space-filling curves, as well as to higher dimensions. It is worth noting the

[1]In the traditional presentation, the two-dimensional Hilbert curve finishes in the bottom right corner.

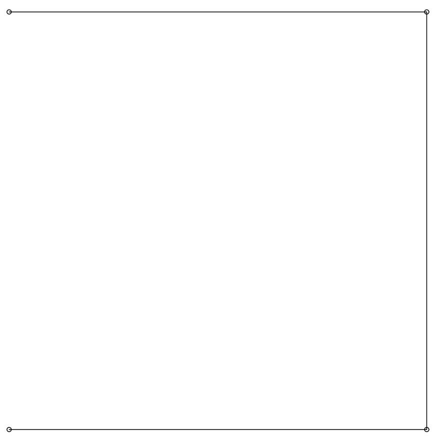


Figure 1: Order 1 Hilbert lattice

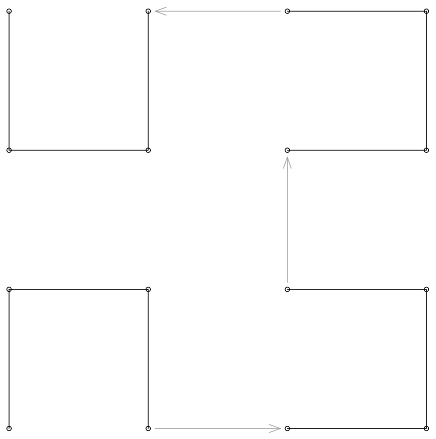


Figure 2: Building the order 2 Hilbert lattice

fact that the Hilbert curves always take steps of unit length: immediate neighbors on the curve are also immediate neighbors in the plane. This translates to a notion of data locality: points close to each other in the plane tend to be close to each other in their associated Hilbert order. For the rest of this section we will be dealing implicitly with Type I Hilbert curves.

2.1 Higher Dimensions

The geometric approach to the two-dimensional Hilbert curve starts by considering a 2×2 lattice of points and describes the path through them. It then recurses by replacing each point with another 2×2 lattice (making a $2^2 \times 2^2$ lattice) and defining the curve through each of those, appropriately rotated such that the entrance and exit points to these sub-lattices remain adjacent. We consider an analogous recursive approach to the multi-dimensional Hilbert curve. Consider a lattice of $2 \times \cdots \times 2$ points in n-dimensions, corresponding to the corners of the unit hypercube in $\mathbb{R}^n$. The key property of the Hilbert curve is that successive points are immediate neighbors in the lattice. Thus, to maintain this property we are looking for a walk through the 2^n points such that every point will be enumerated, and successive points will be neighboring corners of the hypercube.

We let each of the 2^n vertices be labelled by an n-bit string of the form $b = [\beta_{n-1} \cdots \beta_0]_{[2]}$, where $\beta_i \in \mathbb{B}$ represents the position of the vertex along dimension i (0 for low, 1 for high). This is easily interpreted as an n-bit

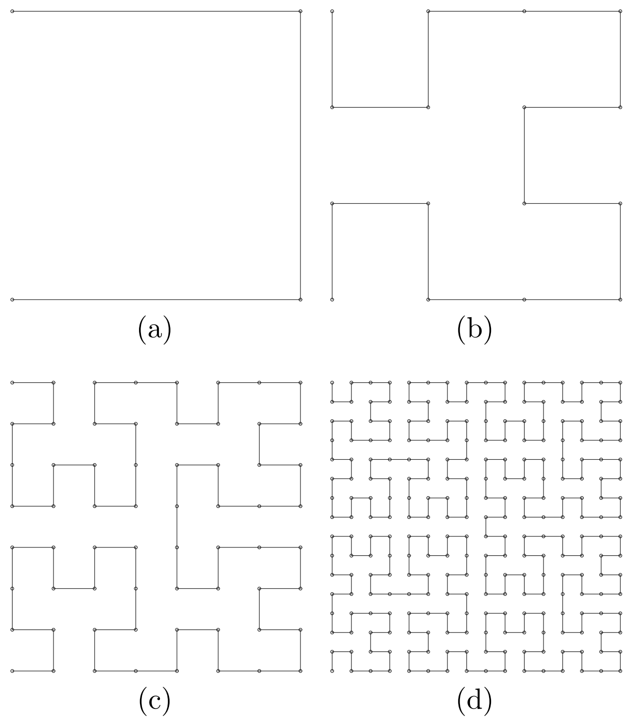


Figure 3: First four iterations of the Hilbert curve.

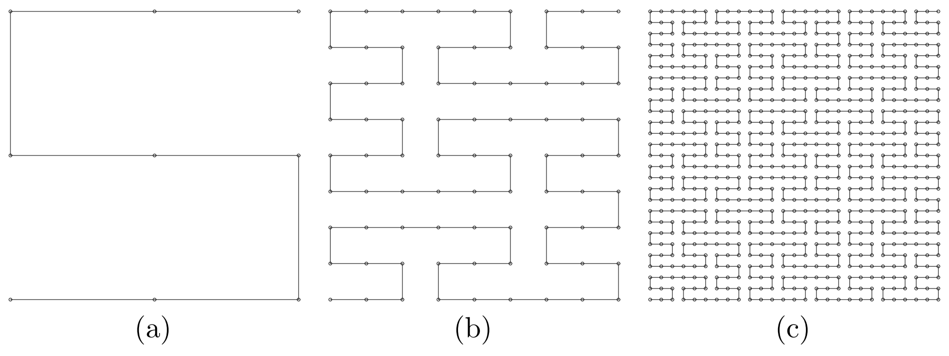


Figure 4: First three iterations of the Peano curve.

non-negative integer value in $\mathbb{Z}_{2^n}$, or equivalently, $\mathbb{B}^n$. Restricting ourselves to taking steps to immediate neighbors implies that in the binary label of successive vertices, only one bit may change. In other words, we are looking for an ordering of the 2^n distinct n-bit numbers such that any successive pair of numbers differ in exactly one bit. This corresponds exactly to the classic Gray code [10].

2.1.1 Gray Code

In general, a Gray code is an ordering of numbers such that adjacent numbers differ in exactly a single digit with respect to some base. More specifically, we are concerned with a binary Gray code. Perhaps the simplest form of a binary Gray code is the binary *reflected* Gray code, which is intuitively constructed in the following manner:

1. Start with the Gray code over 1-bit numbers:
$$[[0]_{[2]}, [1]_{[2]}]$$
2. Write the sequence forwards and then backwards, prepending zeroes to the first half and ones to the second half. This creates the Gray code over all 2-bit numbers:
$$[[00]_{[2]}, [01]_{[2]}, [11]_{[2]}, [10]_{[2]}]$$
3. Repeat step 2, each time growing the Gray code over k-bit numbers to one over $k+1$ bits.

Assuming the input to step 2 is itself a Gray code over all k-bit numbers, it is easy to see that the output will be a valid Gray code over all $(k+1)$-bit numbers. The 2-bit Gray code generated in this manner coincides exactly with the ordering through the four points in a two-dimensional Hilbert curve (it generates the familiar '$\sqsupset$' shape), and it can be used as a basis to extend the concept of the Hilbert curve to higher dimensions[2]. Given this construction of the binary reflected Gray code, we may easily derive a closed form for the ith Gray code integer. The following results on Gray Codes are well known, but we have provided original proofs for the sake of completeness.

[2]This exact agreement is due to the non-standard orientation we have chosen for the Hilbert curve. Given the standard orientation, the agreement would only be up to a rotation.

Theorem 2.1 (Closed-form Binary Reflected Gray Code) *The binary reflected Gray code sequence is generated by the function*

$$\mathrm{gc}(i) = i \veebar (i \triangleright 1).$$

Proof: We consider the value of the jth bit of the ith Gray code, $\mathrm{bit}\,(\mathrm{gc}(i), j)$. The construction begins with the Gray code sequence over $\mathbb{B}$. After j iterations we have the Gray code defined over all $(j+1)$-bit values. In this sequence, the first half of the values are defined such that the jth bit is zero, while the second half has a one for the jth bit. In the next iteration of this construction the pattern reverses itself such that (for $0 \le i < 4(2^j)$):

$$\mathrm{bit}\,(\mathrm{gc}(i), j) = \begin{cases} 0, \text{if } 0 \le i < 2^j, \\ 1, \text{if } 2^j \le i < 2(2^j), \\ 1, \text{if } 2(2^j) \le i < 3(2^j), \\ 0, \text{if } 3(2^j) \le i < 4(2^j) \end{cases} = \begin{cases} 0, & \text{if } \lfloor \frac{i}{2^j} \rfloor = 0, \\ 1, & \text{if } \lfloor \frac{i}{2^j} \rfloor = 1, \\ 1, & \text{if } \lfloor \frac{i}{2^j} \rfloor = 2, \\ 0, & \text{if } \lfloor \frac{i}{2^j} \rfloor = 3. \end{cases}$$

In subsequent iterations of the construction this pattern will simply be repeated as it is already symmetric. Hence, it follows that for *all* $i \ge 0$

$$\mathrm{bit}\,(\mathrm{gc}(i), j) = \begin{cases} 0, & \text{if } \lfloor \frac{i}{2^j} \rfloor \bmod 4 = 0, \\ 1, & \text{if } \lfloor \frac{i}{2^j} \rfloor \bmod 4 = 1, \\ 1, & \text{if } \lfloor \frac{i}{2^j} \rfloor \bmod 4 = 2, \\ 0, & \text{if } \lfloor \frac{i}{2^j} \rfloor \bmod 4 = 3. \end{cases}$$

Since $\lfloor \frac{i}{2^{j+1}} \rfloor \bmod 2 = 1$ if and only if $\lfloor \frac{i}{2^j} \rfloor \bmod 4 \in \{2, 3\}$ we see that

$$\begin{aligned} \mathrm{bit}\,(\mathrm{gc}(i), j) &= \begin{cases} 0, & \text{if } \lfloor \frac{i}{2^j} \rfloor \bmod 4 = 0, \\ 1, & \text{if } \lfloor \frac{i}{2^j} \rfloor \bmod 4 = 1, \\ 0, & \text{if } \lfloor \frac{i}{2^j} \rfloor \bmod 4 = 2, \\ 1, & \text{if } \lfloor \frac{i}{2^j} \rfloor \bmod 4 = 3 \end{cases} + \left\lfloor \frac{i}{2^{j+1}} \right\rfloor \bmod 2 \\ &= \left\lfloor \frac{i}{2^j} \right\rfloor + \left\lfloor \frac{i}{2^{j+1}} \right\rfloor \bmod 2 \\ &= (i \triangleright j) + (i \triangleright (j+1)) \bmod 2 \\ &= \mathrm{bit}\,(i, j) + \mathrm{bit}\,(i, j+1) \bmod 2 \\ &= \mathrm{bit}\,(i, j) \veebar \mathrm{bit}\,(i \triangleright 1, j) \\ &= \mathrm{bit}\,(i \veebar (i \triangleright 1), j)\,. \end{aligned}$$

Thus it follows that $\mathrm{gc}(i) = i \veebar (i \triangleright 1)$. ■

Given a non-negative integer we may wish to find at which position it lies in the Gray code sequence. In other words, we may wish to determine the inverse of the Gray code.

Theorem 2.2 (Binary Reflected Gray Code Inverse) *Consider a non-negative integer i. Let m be the precision of i. That is, let $m = \lceil \log_2(i+1) \rceil$ such that i requires m bits in its binary representation. Then it follows that*

$$\mathrm{bit}\,(i,j) = \sum_{k=j}^{m-1} \mathrm{bit}\,(\mathrm{gc}(i),k)\,.$$

Proof: By Theorem 2.1 we have

$$\mathrm{bit}\,(\mathrm{gc}(i),j) = \mathrm{bit}\,(i,j) + \mathrm{bit}\,(i,j+1) \mod 2.$$

Summing over $j \le k < m$ we find that

$$\begin{aligned}
\sum_{k=j}^{m-1} \mathrm{bit}\,(\mathrm{gc}(i),k) &= \sum_{k=j}^{m-1} \big(\mathrm{bit}\,(i,k) + \mathrm{bit}\,(i,k+1)\big) \mod 2 \\
&= \left(\sum_{k=j}^{m-1} \mathrm{bit}\,(i,k) + \sum_{k=j+1}^{m} \mathrm{bit}\,(i,k) \right) \mod 2 \\
&= \mathrm{bit}\,(i,j) + \left(2 \sum_{k=j+1}^{m-1} \mathrm{bit}\,(\mathrm{gc}(i),k) \right) + \mathrm{bit}\,(i,m) \mod 2 \\
&= \mathrm{bit}\,(i,j) + \mathrm{bit}\,(i,m) \mod 2.
\end{aligned}$$

By the definition of m we see that $\mathrm{bit}\,(i,m) = 0$ and the result follows. ■

We may use Theorem 2.2 to construct Algorithm 1, which computes the inverse as desired.

We are interested in knowing along which bit the Gray code will change when preceding from one term to the next. Equivalently, we are interested in knowing along which dimension we will step when proceeding from one vertex to another on the Hilbert curve. To this end, we define $\mathrm{g}(i)$ as

$$\mathrm{g}(i) = k, \text{ such that } \mathrm{gc}(i) \veebar \mathrm{gc}(i+1) = 2^k, \quad 0 \le i < 2^n - 1.$$

Lemma 2.3 (Dimension of Change in the Gray Code) *The sequence $\mathrm{g}(i)$ is given by*

$$\mathrm{g}(i) = \mathrm{tsb}(i),$$

where tsb *is the number of* trailing set bits *in the binary representation of i.*

Algorithm 1 GrayCodeInverse(g)

Given a non-negative integer g, calculates the non-negative integer i such that $\mathrm{gc}(i) = g$.

Input: A non-negative integer g.

Output: The non-negative integer i such that $\mathrm{gc}(i) = g$.

```
1: $m \leftarrow$ number of bits required to represent $g$
2: $(i, j) \leftarrow (g, 1)$
3: while $j < m$ do
4:     $i \leftarrow i \veebar (g \triangleright j)$
5:     $j \leftarrow j + 1$
6: end while
```

Proof: We examine the difference between two consecutive values of the Gray code:

$$\begin{aligned}\mathrm{gc}(i) \veebar \mathrm{gc}(i+1) &= i \veebar (i \triangleright 1) \veebar (i+1) \veebar ((i+1) \triangleright 1) \\ &= (i \veebar (i+1)) \veebar ((i \triangleright 1) \veebar ((i+1) \triangleright 1)) \\ &= (i \veebar (i+1)) \veebar ((i \veebar (i+1)) \triangleright 1).\end{aligned}$$

We consider first the portion $i \veebar (i+1)$. Adding 1 to i will cause a carry past the first digit if the first digit is 1. Similarly past the second digit and so on. Letting k be the number of trailing ones in the binary representation of i (or, alternatively, the index of the first zero valued bit), it follows that $i + 1$ will have a one at position k, zeroes at positions 0 through $k - 1$, and be identical to i elsewhere. Thus, taking the exclusive-or of these two will result in a number with $k + 1$ trailing one bits. Similarly, the result of $(i \veebar (i + 1)) \triangleright 1$ will be a number with k trailing one bits. Taking the exclusive-or of these two results in a number with a single non-zero bit at the kth position. Hence, between the ith and $(i + 1)$th Gray code integers it is the kth bit that changes. This corresponds exactly to the definition of 'tsb' thus it follows that $\mathrm{g}(i) = \mathrm{tsb}(i)$. ■

Lemma 2.4 (Symmetry of the Gray Code) *Given $n \in \mathbb{N}$ and $0 \leq i < 2^n$, it follows that* $\mathrm{gc}(2^n - 1 - i) = \mathrm{gc}(i) \veebar 2^{n-1}$.

Proof: This property follows immediately from the construction algorithm for the reflected binary Gray code. The second 2^{n-1} values are simply equal to the first 2^{n-1} values in reverse, with the $(n - 1)$th bit set as a 1. Thus we see that

$$\mathrm{gc}(2^n - 1 - i) = \mathrm{gc}(i) \vee 2^{n-1}, \text{ for } 0 \leq i < 2^{n-1}.$$

Replacing the 'or' operation by an 'exclusive-or' (justified in this case as exaclty one of $\mathrm{gc}(2^n-1-i)$ or $\mathrm{gc}(i)$ will have a zero in the $(n-1)$th bit) leads to the desired result. ■

Corollary 2.5 (Symmetry of $\mathrm{g}(i)$) *The sequence* $\mathrm{g}(i)$ *is symmetric such that* $\mathrm{g}(i) = \mathrm{g}(2^n-2-i)$ *for* $0 \le i \le 2^n-2$.

Proof: Without loss of generality we consider $i \le \frac{2^n-2}{2}$. Lemma 2.4 tells us that

$$\mathrm{gc}(2^n-2-i) = \mathrm{gc}(i+1) \veebar 2^{n-1}.$$

By the definition of gc we know that $\mathrm{gc}(i+1) = \mathrm{gc}(i) \veebar 2^{\mathrm{g}(i)}$ and $\mathrm{gc}(2^n-2-i) = \mathrm{gc}(2^n-1-i) \veebar 2^{\mathrm{g}(2^n-2-i)}$. Substituting these into the above equation yields

$$\mathrm{gc}(2^n-1-i) \veebar 2^{\mathrm{g}(2^n-2-i)} = \mathrm{gc}(i) \veebar 2^{\mathrm{g}(i)} \veebar 2^{n-1}.$$

By Lemma 2.4 this simplifies to the desired result,

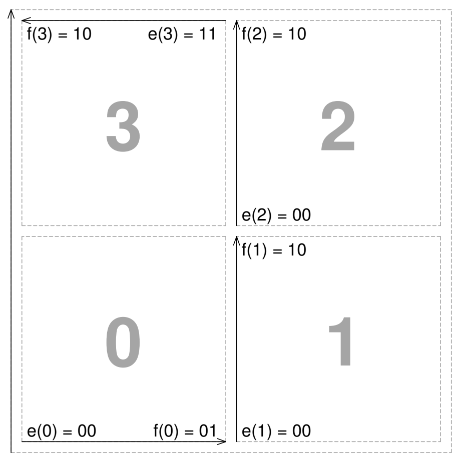
$$\mathrm{g}(2^n-2-i) = \mathrm{g}(i).$$

■

Analogous to the Hilbert curve in two-dimensions, the Gray code ordering can be used to give an ordering through the vertices of a unit hypercube in $\mathbb{R}^n$. As in the recursive construction in two dimensions, we will recursively define the Hilbert curve by zooming in on each point in the sequence (each sub-hypercube) and iterating through the points within using a transformed/rotated version of the original curve. Like the two-dimensional case, we must determine orientations for the Hilbert curve through each of the 2^n sub-hypercubes. These orientations must be consistent in that the exit point of the curve through one sub-hypercube must be immediately adjacent to the entry point of the next sub-hypercube. Additionally, the entry and exit points of the parent hypercube must coincide with the entry point of the first sub-hypercube and the exit point of the last sub-hypercube, respectively. These constraints on entry and exit points are visualized for the two-dimensional case in Figure 5.

2.1.2 Entry Points

Using the same labelling as the vertices of the parent hypercube, we let $\mathrm{e}(i)$ and $\mathrm{f}(i)$ refer, respectively, to the entry and exit vertices of the ith sub-hypercube in a Gray code ordering of the sub-hypercubes. Since the ith



i	$\mathrm{e}(i)$	$\mathrm{f}(i)$	$\mathrm{d}(i)$	$\mathrm{g}(i)$
0	$[00]_{[2]}$	$[01]_{[2]}$	0	0
1	$[00]_{[2]}$	$[10]_{[2]}$	1	1
2	$[00]_{[2]}$	$[10]_{[2]}$	1	0
3	$[11]_{[2]}$	$[10]_{[2]}$	0	–

Figure 5: Entry and exit points of the 2 dimensional Hilbert curve (the x-axis corresponds to the least significant bit and the y-axis the most significant).

and $(i+1)$th sub-hypercubes are neighbors along the $\mathrm{g}(i)$th coordinate, we must have that $\mathrm{f}(i) \veebar 2^{\mathrm{g}(i)} = \mathrm{e}(i+1)$. Like the entry and exit points of the parent hypercube, entry and exit points of a given sub-hypercube must be neighboring corners. That is, $\mathrm{e}(i)$ and $\mathrm{f}(i)$ may only differ in exactly one bit position, meaning we must have that $\mathrm{e}(i) \veebar \mathrm{f}(i) = 2^{\mathrm{d}(i)}$ for some $\mathrm{d}(i) \in \mathbb{Z}_n$. We refer to $\mathrm{d}(i)$ as the *intra* sub-hypercube direction, and $\mathrm{g}(i)$ as the *inter* sub-hypercube direction. Combining these two results shows that entry points must satisfy the relation

$$\mathrm{e}(i+1) = \mathrm{e}(i) \veebar 2^{\mathrm{d}(i)} \veebar 2^{\mathrm{g}(i)}, \quad 0 \le i < 2^n - 1. \tag{1}$$

Additionally, as mentioned earlier, we must have that $\mathrm{e}(0)$ is the same as the entry point of the parent hypercube and $f(2^n - 1)$ is the same as the exit point of the parent hypercube. These constraints are displayed graphically for the two-dimensional case in Figure 5.

In order to fully determine closed forms for $\mathrm{e}(i)$, $\mathrm{d}(i)$ and $\mathrm{f}(i)$, we first explore various properties of these sequences.

Lemma 2.6 (Symmetry of $\mathrm{e}(i)$ and $\mathrm{f}(i)$) *The sequences $\mathrm{e}(i)$ and $\mathrm{f}(i)$ are symmetric such that $\mathrm{e}(i) = \mathrm{f}(2^n - 1 - i) \veebar 2^{n-1}$.*

Proof: We consider walking through the Hilbert curve backwards, such that $\bar{e}_i = \mathrm{f}(2^n - 1 - i)$ and the ith sub-hypercube is $\overline{\mathrm{gc}}(i) = \mathrm{gc}(2^n - 1 - i)$. By Lemma 2.4 this is equivalent to $\overline{\mathrm{gc}}(i) = \mathrm{gc}(i) \veebar 2^{n-1}$. Thus, it follows that $\mathrm{f}(2^n - 1 - i) = \bar{e}_i = \mathrm{e}(i) \veebar 2^{n-1}$. ■

Corollary 2.7 (Symmetry of $\mathrm{d}(i)$) *The sequence* $\mathrm{d}(i)$ *is symmetric such that* $\mathrm{d}(i) = \mathrm{d}(2^n - 1 - i)$ *for* $0 \le i \le 2^n - 1$.

Proof: Lemma 2.6 tells us that $\mathrm{e}(i) = \mathrm{f}(2^n - 1 - i) \veebar 2^{n-1}$ and equivalently $\mathrm{e}(2^n - 1 - i) = \mathrm{f}(i) \veebar 2^{n-1}$. Combining these two yields

$$\mathrm{e}(i) \veebar \mathrm{f}(i) = \mathrm{e}(2^n - 1 - i) \veebar \mathrm{f}(2^n - 1 - i).$$

By the definition of $\mathrm{d}(i)$ we have that $\mathrm{e}(i) \veebar \mathrm{d}(i) = \mathrm{f}(i)$ thus we see

$$\mathrm{d}(i) = \mathrm{d}(2^n - 1 - i).$$

■

Lemma 2.8 *Suppose that*

$$\mathrm{d}(i) = \begin{cases} 0, & i = 0; \\ \mathrm{g}(i-1) \bmod n, & i = 0 \bmod 2; \\ \mathrm{g}(i) \bmod n, & i = 1 \bmod 2, \end{cases}$$

for $0 \le i \le 2^n - 1$. *Then* $\mathrm{d}(i)$ *is symmetric as per Corollary 2.7.*

Proof: Suppose $i = 0$. Then $\mathrm{d}(0) = 0$. Similarly, $\mathrm{d}(2^n - 1) = \mathrm{g}(2^n - 1) = \mathrm{tsb}\, 2^n - 1 = n \bmod n = 0$. Suppose $i = 0 \bmod 2$. Then $\mathrm{d}(i) = \mathrm{g}(i-1)$. Since $2^n - 1 - i = 1 \bmod 2$, we see that $\mathrm{d}(2^n - 1 - i) = \mathrm{g}(2^n - 1 - i) = \mathrm{g}(2^n - 2 - (i-1)) = \mathrm{g}(i-1)$. Suppose $i = 1 \bmod 2$. Then $\mathrm{d}(i) = \mathrm{g}(i)$. Since $2^n - 1 - i = 0 \bmod 2$, we see that $\mathrm{d}(2^n - 1 - i) = \mathrm{g}(2^n - 2 - i) = \mathrm{g}(i)$. Thus, this form for $\mathrm{d}(i)$ meets the symmetry requirement of Corollary 2.7. ■

Theorem 2.9 (Intra Sub-hypercube Directions) *The formula of Lemma 2.8 satisfies Equation 1, and hence defines the sequence of intra sub-hypercube directions,* $\mathrm{d}(i)$.

Proof: Let $\mathrm{d}(i, n)$ be the sequence of intra sub-hypercube directions for a fixed dimension n. By inspection (see Figure 5) we see that the above definition holds for the case $n = 2$. Suppose that the definition holds for $1, \dots, n$ and consider the case $n+1$. As long as $\mathrm{g}(i) < n$, then $\mathrm{g}(i) \bmod n+$

$1 = \mathrm{g}(i) \bmod n$. Thus, we consider the first i such that $\mathrm{g}(i) \geq n$. By Lemma 2.3 we see that this occurs when $i = 2^n - 1$, the smallest positive integer with n trailing set bits. Hence, for $i < 2^n - 1$ we must have that $\mathrm{d}(i, n+1) = \mathrm{d}(i, n)$. Now consider $\mathrm{d}(2^n - 1, n+1)$. Since the exit point of the ith cell must touch the face of the $(i+1)$th cell along the $\mathrm{g}(i)$th axis, we must have that

$$\mathrm{bit}\left(\mathrm{f}(i), \mathrm{g}(i)\right) = \mathrm{bit}\left(\mathrm{gc}(i+1), \mathrm{g}(i)\right).$$

Substituting Equation 1 into this we must have that

$$\mathrm{bit}\left(\underline{\bigvee}_{j=0}^{2^n-1} 2^{\mathrm{d}(j,n+1)} \veebar \underline{\bigvee}_{j=0}^{2^n-2} 2^{\mathrm{g}(j)}, \mathrm{g}(2^n-1)\right) = \mathrm{bit}\left(\underline{\bigvee}_{j=0}^{2^n-1} 2^{\mathrm{g}(j)}, \mathrm{g}(2^n-1)\right),$$

which simplifies to

$$\begin{aligned}
\mathrm{bit}\left(\underline{\bigvee}_{j=0}^{2^n-1} 2^{\mathrm{d}(j,n+1)}, \mathrm{g}(2^n-1)\right) &= 1, \\
\mathrm{bit}\left(\underline{\bigvee}_{j=0}^{2^n-2} 2^{\mathrm{d}(j,n+1)} \veebar 2^{\mathrm{d}(2^n-1,n+1)}, \mathrm{g}(2^n-1)\right) &= 1, \\
\mathrm{bit}\left(\underline{\bigvee}_{j=0}^{2^n-2} 2^{\mathrm{d}(j,n)} \veebar 2^{\mathrm{d}(2^n-1,n+1)}, \mathrm{g}(2^n-1)\right) &= 1.
\end{aligned}$$

By the symmetry of $\mathrm{d}(i, n)$ most of the first term cancels, leaving

$$\mathrm{bit}\left(2^{\mathrm{d}(0,n)} \veebar 2^{\mathrm{d}(2^n-1,n+1)}, \mathrm{g}(2^n-1)\right) = 1.$$

Since $\mathrm{d}(0, n) = 0$ then we must have that $\mathrm{d}(2^n - 1, n+1) = \mathrm{g}(2^n - 1)$. We know that $\mathrm{d}(i, n+1)$ holds for $0 \leq 0 \leq 2^n - 1$, and by Lemma 2.8 we know that this holds for the other half, $2^n \leq i \leq 2^{n+1} - 1$. Hence that definition holds for the case $n+1$ and by the inductive hypothesis it holds for all $n \geq 2$. ■

Theorem 2.10 (Entry Points) *The sequence of entry points is defined by*

$$\mathrm{e}(i) = \begin{cases} 0, & i = 0, \\ \mathrm{gc}(2\lfloor \frac{i-1}{2} \rfloor), & 0 < i \leq 2^n - 1. \end{cases}$$

Proof: By recursive application of Equation 1 we have that

$$\mathrm{e}(i) = \bigveebar_{j=0}^{i-1} 2^{\mathrm{d}(j)} \veebar \bigveebar_{j=0}^{i-1} 2^{\mathrm{g}(j)}.$$

By definition, for all n we have that $e(0) = 0$, thus we consider only the case $i > 0$. Simplifying the above yields

$$\begin{aligned} \mathrm{e}(i) &= 2^{\mathrm{g}(0)} \veebar \bigveebar_{j=1}^{i-1} 2^{\mathrm{d}(j)} \veebar \mathrm{gc}(i) \\ &= 2^{\mathrm{g}(0)} \veebar \underbrace{2^{\mathrm{d}(0)} \veebar 2^{\mathrm{d}(1)}} \veebar \underbrace{2^{\mathrm{d}(2)} \veebar 2^{\mathrm{d}(3)}} \veebar \ldots \veebar 2^{\mathrm{d}(i-1)} \veebar \mathrm{gc}(i). \end{aligned}$$

Suppose $i = 0 \bmod 2$. Then by Theorem 2.9 all of the $\mathrm{d}(i)$ cancel out except $\mathrm{d}(i-1)$, leaving us with $\mathrm{e}(i) = 2^{\mathrm{g}(0)} \veebar 2^{\mathrm{d}(i-1)} \veebar \mathrm{gc}(i)$. Since $\mathrm{g}(0) = \mathrm{tsb}(0) = 0 = \mathrm{tsb}(i) = \mathrm{g}(i)$ this yields $\mathrm{e}(i) = \mathrm{gc}(i) \veebar 2^{\mathrm{d}(i-1)} \veebar 2^{\mathrm{g}(i)} = \mathrm{gc}(i) \veebar 2^{\mathrm{g}(i-1)} \veebar 2^{\mathrm{g}(i)} = \mathrm{gc}(i-2)$. Thus, $\mathrm{e}(i) = \mathrm{gc}(2\lfloor \frac{i-1}{2} \rfloor)$.

Suppose now that $i = 1 \bmod 2$. All of the $\mathrm{d}(i)$ cancel, leaving $\mathrm{e}(i) = \mathrm{gc}(i) \veebar 2^{\mathrm{g}(0)}$. Since $\mathrm{g}(0) = \mathrm{tsb}(0) = 0 = \mathrm{tsb}\, i - 1 = \mathrm{g}(i-1)$ this simplifies to $\mathrm{e}(i) = \mathrm{gc}(i-1)$. For $i = 1 \bmod 2$ we have that $i - 1 = 2\lfloor \frac{i-1}{2} \rfloor$, hence $\mathrm{e}(i) = \mathrm{gc}(2\lfloor \frac{i-1}{2} \rfloor)$. ■

2.1.3 Rotations and Reflections

As noted in Section 2.1.1 the recursive construction of the Hilbert curve requires us to construct a curve through the corners of a hypercube when provided with a particular entry and exit point. The classic Gray code explored earlier starts at $\mathrm{gc}(0) = 0$ and ends at $\mathrm{gc}(2^n - 1) = 2^{n-1}$, thus implicitly has an entry point $e = 0$, an internal direction $d = n - 1$ and an exit point $f = 2^{n-1}$. We wish to define a geometric transformation such that the Gray code ordering of sub-hypercubes in the Hilbert curve defined by e and d will map to the standard binary reflected Gray code.

To this end, let us define the *right bit rotation* operator $\circlearrowright$ as

$$b \circlearrowright i = \left[b_{(n-1+i \bmod n)} \cdots b_{(i \bmod n)}\right]_{[2]}, \text{ where } b = [b_{n-1} \cdots b_0]_{[2]}.$$

Conceptually, this function rotates the n bits of b to the right by i places. Analogously, we define the *left bit rotation* operator, $\circlearrowleft$. Trivially, both the left and right bit rotation operators are bijective over $\mathbb{Z}_2^n$ (or equivalently $\mathbb{B}^n$) for any given i. Given e and d, we may now define a transformation T as

$$T_{(e,d)}(b) = (b \veebar e) \circlearrowright (d+1).$$

Being the composition of two bijective operators, we see that the mapping is itself bijective. We first explore the behaviour of the mapping on the entry and exit points.

Lemma 2.11 (Transformed Entry and Exit Points) *The transform $T_{(e,d)}$ maps e and f to the first and last terms, respectively, of the binary reflected Gray code sequence over $\mathbb{B}^n$. That is,*

$$T_{(e,d)}(e) = 0, \text{ and } T_{(e,d)}(f) = 2^{n-1}.$$

Proof: Straightforward:

$$\begin{aligned}
T_{(e,d)}(e) &= (e \veebar e) \circlearrowleft (d+1) = 0 \circlearrowleft d+1 = 0; \text{ and,} \\
T_{(e,d)}(f) &= (f \veebar e) \circlearrowleft (d+1) \\
&= (e \veebar 2^d \veebar e) \circlearrowleft (d+1) \\
&= 2^d \circlearrowleft (d+1) \\
&= \left[\underbrace{0\cdots0}_{n-d-1}\ 1\underbrace{0\cdots0}_{d}\right]_{[2]} \circlearrowleft (d+1) \\
&= \left[1\underbrace{0\cdots0}_{n-1}\right]_{[2]} = 2^{n-1}.
\end{aligned}$$

■

Given the nature of bit-rotation and the exclusive-or operator, it is also easy to see that if neighboring elements of a sequence differ in only one bit position, then the same will hold true for the two transformed points. Hence, they will be neighbors as well. This and the fact that the mapping is bijective tells us that T will in fact preserve this critical property of a Gray code sequence.

Lemma 2.12 (Inverse Transform) *The inverse of the transform $T_{(e,d)}$ is itself a T-transform, given by*

$$T^{-1}_{(e,d)} = T_{(e \circlearrowleft (d+1), n-d-1)}.$$

Proof: It is easy to see that $(T_{(e,d)}(a) \circlearrowleft (d+1)) \veebar e = a$, simply by reversing the individual operations of $T_{(e,d)}$. Letting $b = T_{(e,d)}(a)$, this simplifies to

$$\begin{aligned}
(b \circlearrowleft (d+1)) \veebar e &= (b \circlearrowright (n-d-1)) \veebar e \\
&= \big(b \circlearrowright (n-d-1)\big) \veebar \big(e \circlearrowleft (n-d-1) \circlearrowright (n-d-1)\big) \\
&= \big(b \veebar (e \circlearrowleft (n-d-1))\big) \circlearrowright (n-d-1) \\
&= \big(b \veebar (e \circlearrowright (d+1)\big) \circlearrowright (n-d-1).
\end{aligned}$$

■

We are now ready to construct the Hilbert curve starting at e with direction d. We define $\mathrm{gc}_{(e,d)}(i) = T^{-1}_{(e,d)}(\mathrm{gc}(i))$. By our earlier discussion it follows that the sequence generated by $\mathrm{gc}_{(e,d)}$ is a Gray code sequence. Furthermore, by Lemmas 2.11 and 2.12 it follows that this Gray code sequence begins and ends on the desired points, and the mapping $T_{(e,d)}$ maps it back to the standard binary reflective Gray code. Hence, we now have the tools necessary to consistently construct Hilbert curves through hypercubes with arbitrarily defined entry points and directions. We finish this section with one last result on composed transforms which will be necessary later to deal with the recursive nature of the Hilbert curve.

Lemma 2.13 (Composed Transforms) *Consider the composed transform*

$$b = T_{(e_2,d_2)}\big(T_{(e_1,d_1)}(a)\big).$$

Then it follows that

$$b = T_{(e,d)}(a)$$

where $e = e_1 \veebar (e_2 \circlearrowleft (d_1+1))$ *and* $d = d_1 + d_2 + 1$.

Proof: Straightforward:

$$\begin{aligned}
T_{(e_2,d_2)}\big(T_{(e_1,d_1)}(a)\big) &= T_{(e_2,d_2)}\big((a \veebar e_1) \circlearrowright (d_1+1)\big) \\
&= T_{(e_2,d_2)}\Big(\big(a \circlearrowright (d_1+1)\big) \veebar \big(e_1 \circlearrowright (d_1+1)\big)\Big) \\
&= \Big(\big(a \circlearrowright (d_1+1)\big) \veebar \big(e_1 \circlearrowright (d_1+1)\big) \veebar e_2\Big) \circlearrowright (d_2+1) \\
&= \Big(a \veebar \underbrace{e_1 \veebar \big(e_2 \circlearrowleft (d_1+1)\big)}_{e}\Big) \circlearrowright (\underbrace{d_1+d_2+1}_{d}+1).
\end{aligned}$$

■

It is important to note that T-transforms do in fact have the desired geometric interpretation when applied to our binary labels of the vertices of the unit hypercube. The bit rotation operator can be interpreted as a rotation operator in $\mathbb{R}^n$, while the exclusive-or operation can be interpreted as a mirroring operation, inverting the axes i where $\operatorname{bit}(e, i) = 1$. Hence, the T-transform may be interpreted as a type of rotation and reflection operator over the space $\mathbb{R}^n$.

2.1.4 Algorithms

We consider a space of n-dimensional vectors where each component is an integer of precision m; that is, where each component may be represented using m bits. Given the Hilbert curve through this space $\mathbb{B}^n_m$, we wish to determine the Hilbert index, h, of a given point $\mathbf{p} = [p_0, \dots, p_{n-1}], p_i \in \mathbb{B}^m$.

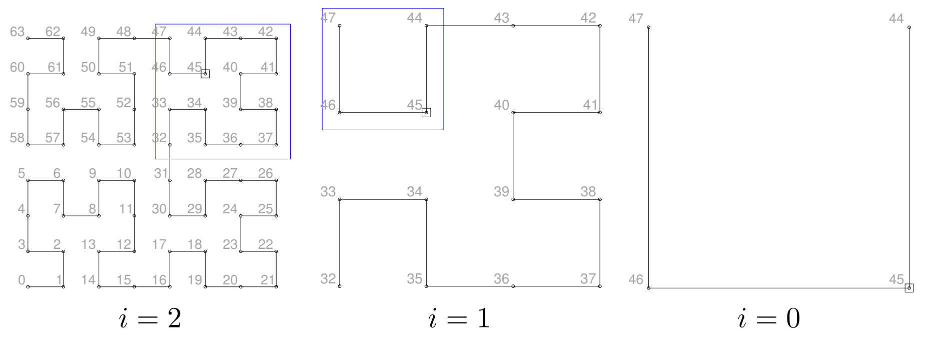
The result may be found in a series of m projections and Gray code calculations. Given $\mathbf{p}$, we may extract an n-bit number

$$l_{m-1} = [\operatorname{bit}(p_{n-1}, m-1) \cdots \operatorname{bit}(p_0, m-1)]_{[2]}\,.$$

Each bit of l tells us whether the point $\mathbf{p}$ is in the lower or upper half set of points with respect to a given axis. Thus, the point l_{m-1} locates in which sub-hypercube the point $\mathbf{p}$ may be found. Equivalently, it tells us the vertex of the Hilbert curve through the vertices of the unit hypercube to which p belongs. We wish to determine the Hilbert index of the sub-hypercube containing $\mathbf{p}$, given e and d. As discussed in Section 2.1.3, we do this in two steps: (1) rotate and reflect the space such that the Gray code ordering corresponds to the binary reflected Gray code, $\bar{l}_{m-1} = T_{(e,d)}(l_{m-1})$; and, (2) determine the index of the associated sub-hypercube, $w_{m-1} = \operatorname{gc}^{-1}(\bar{l}_{m-1})$.

We may now calculate $\operatorname{e}(w_{m-1})$ and $\operatorname{d}(w_{m-1})$ in order to determine the entry point and direction through the sub-hypercube containing the point $\mathbf{p}$. The values $\operatorname{e}(w_{m-1})$ and $\operatorname{d}(w_{m-1})$ are relative to the transformed space, thus we may compose this transformation with the existing transformation using Lemma 2.13, calculating $e = e \veebar (\operatorname{e}(w_{m-1}) \circlearrowleft (d+1))$ and $d = d + \operatorname{d}(w_{m-1}) + 1$. At this point, the parameters e and d describe the rotation and reflection necessary to map the sub-hypercube containing $\mathbf{p}$ back to the standard orientation. We then narrow our focus on the sub-hypercube containing $\mathbf{p}$. We repeat the above steps to calculate w_{m-2} and update e and d appropriately. We continue for w_{m-3} through w_0 and finally calculate the full Hilbert index as

$$h = [w_{m-1} w_{m-2} \cdots w_0]_{[2]} = \sum_{i=0}^{m-1} 2^{ni} w_i = \bigvee_{i=0}^{m-1} (w_i \triangleleft ni).$$



$$\mathbf{p} = [5, 6] = \left[[101]_{[2]}, [110]_{[2]}\right]$$

i	l	$T_{(e,d)}(l)$	w	e(w)	d(w)	e	d	h
-	-	-	-	-	-	0	1	0
2	$[11]_{[2]} = 3$	3	2	0	1	0	1	2
1	$[10]_{[2]} = 2$	2	3	3	0	3	0	11
0	$[01]_{[2]} = 1$	1	1	0	1	3	0	45

Figure 6: Running algorithm HilbertIndex with $n = 2$, $m = 3$ and $\mathbf{p} = [5, 6]$.

We formalize this approach in Algorithm 2. Inverting the HilbertIndex algorithm is straightforward, with the inverse given by Algorithm 3[3].

We consider an example in two dimensions where $m = 3$. Let $\mathbf{p}$ be the point at $x = 5$ and $y = 6$. Figure 6 displays both graphically and in tabular form the results of running Algorithm 2 on the point $\mathbf{p}$.

3 Compact Hilbert Indices

As mentioned in Section 1, one use of Hilbert curves is in the segmentation of multidimensional data for R-trees [15]. Hilbert curves prove to be excellent at preserving data locality and minimizing data fragmentation as

[3]In these algorithms we have chosen to initialize the direction d as 0, instead of $n - 1$. This corresponds to the standard orientation where the Hilbert curve starts and finishes at opposite ends of the x-axis, the orientation we had originally shunned in Section 1.

Algorithm 2 $\textsc{HilbertIndex}(n, m, \mathbf{p})$

Calculates the Hilbert index $h \in \mathbb{B}^{mn}$ of a point $\mathbf{p} \in \mathbb{P}$.

Input: $n, m \in \mathbb{Z}_+$ and a point $\mathbf{p} \in \mathbb{P}$.

Output: $h \in \mathbb{B}^M$, the Hilbert index of the point $\mathbf{p}$.

```
1: $(h, e, d) \leftarrow (0, 0, 0)$
2: for $i = m - 1$ to 0 do
3:     $l \leftarrow [\mathrm{bit}\,(p_{n-1}, i) \cdots \mathrm{bit}\,(p_0, i)]_{[2]}$
4:     $l \leftarrow T_{(e,d)}(l)$
5:     $w = \mathrm{gc}^{-1}(l)$
6:     $e \leftarrow e \veebar (\mathrm{e}(w) \circlearrowleft (d + 1))$
7:     $d \leftarrow d + \mathrm{d}(w) + 1 \bmod n$
8:     $h \leftarrow (h \triangleleft n) \vee w$
9: end for
```

Algorithm 3 $\textsc{HilbertIndexInverse}(n, m, h)$

Calculates the point $\mathbf{p} \in \mathbb{P}$ corresponding to a given Hilbert index $h \in \mathbb{B}^{mn}$.

Input: $n, m \in \mathbb{Z}_+$ and $h \in \mathbb{B}^{mn}$, the Hilbert index of the point $\mathbf{p}$.

Output: A point $\mathbf{p} \in \mathbb{P}$.

```
1: $(e, d) \leftarrow (0, 0)$
2: $\mathbf{p} = [p_0, \ldots, p_{n-1}] \leftarrow [0, \ldots, 0]$
3: for $i = m - 1$ to 0 do
4:     $w \leftarrow [\mathrm{bit}\,(h, in + n - 1) \cdots \mathrm{bit}\,(h, in + 0)]_{[2]}$
5:     $l = \mathrm{gc}(w)$
6:     $l \leftarrow T^{-1}_{(e,d)}(l)$
7:     for $j = 0$ to $n - 1$ do
8:         $\mathrm{bit}\,(p_j, i) \leftarrow \mathrm{bit}\,(l, j)$
9:     end for
10:    $e \leftarrow e \veebar (\mathrm{e}(w) \circlearrowleft (d + 1))$
11:    $d \leftarrow d + \mathrm{d}(w) + 1 \bmod n$
12: end for
```

compared to other space-filling curves [21]. In this setting, large data-sets of multidimensional data are sorted based on their Hilbert index. Most real-world applications have some need to perform operations on the points with respect to their Hilbert indices, including merging lists of Hilbert sorted points. In these cases it is often convenient or even preferable to store the Hilbert index of the point so as to work directly with it. Unfortunately, in most cases not all of the dimensions are of the same size thus the Hilbert index, forced to be defined over a hypercube where all dimensions are the same size, may be much larger than the original data. This costs space when storing Hilbert indices and time when comparing them. It would be desirable to construct an ordering on the points that requires an index of the same size of the incoming data.

Consider an n-dimensional data set consisting of points $\mathbf{p} \in \mathbb{B}^{m_0} \times \cdots \times \mathbb{B}^{m_{n-1}} = \mathbb{P}$, where $m_i \in Z_+$ is the precision of the data in the ith dimension. Storing a point of data requires $M = \sum_i m_i$ bits. However, a Hilbert index must be calculated with respect to a hypercube of precision $m = \max_i\{m_i\}$, and requires $mn \geq M$ bits of storage. As an example, we consider a customer database containing an *id*, a *province* and a *gender* of 16, 4 and 1 bits respectively. Points in their native space require $16 + 4 + 1 = 21$ bits to store, while the associated Hilbert indices will require $3 \times 16 = 48$ bits, representing a data expansion factor of $48/21 \approx 2.29$.

We wish to find an indexing scheme that preserves completely the ordering of the Hilbert indices, but requires only M bits to represent. A simple method to do this is to walk through all the points in $\mathbb{P}$, calculate their Hilbert indices and sort them based on these Hilbert indices. Then, assign to each point its rank as an index. Trivially, this index has the same ordering as the Hilbert ordering over $\mathbb{P}$, and it requires only $\sum_i m_i$ bits to represent. However, in order to generate such an index we must first enumerate the entire space, a prohibitive cost. The key to calculating this index directly, referred to as the *compact Hilbert index*, lies in a simple observation about Gray Codes.

3.1 Gray Code Rankings

We consider a Gray code $\text{gc}(i)$, where $\text{gc}(i)$ is an n-bit integer where some subset of the bits are fixed. We let μ be a *mask* and π be a *pattern* such that $\pi \wedge \mu = 0$. We restrict ourselves to values $\text{gc}(i)$ where $\text{bit}(\text{gc}(i), j) = \text{bit}(\pi, j)$ when $\text{bit}(\mu, j) = 0$. This is equivalent to restricting ourselves to values $\text{gc}(i)$ such that $\text{gc}(i) \wedge !\mu = \pi$. We let $\mathcal{I}$ be the set of integers that satisfy this condition, $\mathcal{I} = \{i |\, \text{gc}(i) \wedge !\mu = \pi\}$. Since $\|\mu\|$ counts the number of unconstrained bits in the definition of $\mathcal{I}$, it is easy to see that $|\mathcal{I}| = 2^{\|\mu\|} \leq 2^n$. We wish to determine a $\|\mu\|$-bit value, the *Gray code rank*, such that for all $i \neq j \in \mathcal{I}$, $i < j$ if and only if $\text{gcr}(i) < \text{gcr}(j)$. It is plain to see that $\text{gcr}(i)$ must be equal to the rank of i with respect to all entries in $\mathcal{I}$. However, we wish to calculate the rank directly without having to enumerate over the entire set $\mathcal{I}$.

$\text{gc}(i)$	8	10	12	14	20	26	28	30
i	15	12	8	11	16	19	23	20
$\text{gcr}(i)$	3	2	0	1	4	5	7	6
$[\text{gc}(i)]_{[2]}$	$0\underline{0}1\underline{00}0$	$0\underline{0}1\underline{01}0$	$0\underline{0}1\underline{10}0$	$0\underline{0}1\underline{11}0$	$0\underline{1}1\underline{00}0$	$0\underline{1}1\underline{01}0$	$0\underline{1}1\underline{10}0$	$0\underline{1}1\underline{11}0$
$[i]_{[2]}$	$0\underline{0}1\underline{11}1$	$0\underline{0}1\underline{10}0$	$0\underline{0}1\underline{00}0$	$0\underline{0}1\underline{01}1$	$0\underline{1}0\underline{00}0$	$0\underline{1}0\underline{01}1$	$0\underline{1}0\underline{11}1$	$0\underline{1}0\underline{10}0$
$[\text{gcr}(i)]_{[2]}$	011	010	000	001	100	101	111	110

Table 1: Values of $\text{gc}(i)$, i and $\text{gcr}(i)$ for $\mu = [010110]_{[2]}$ and $\pi = [001000]_{[2]}$.

We consider an example where $n = 6$, $\|\mu\| = 3$, $\mu = [010110]_{[2]}$ and $\pi = [001000]_{[2]}$, shown in Table 1. The unconstrained bits are shown underlined to help in visualizing the effect of the mask and pattern. With a quick visual inspection it becomes readily apparent that the $\text{gcr}(i)$ values can be constructed simply by concatenating the unconstrained bits from i. We formalize this concept with the following results.

Lemma 3.1 (Principal Bits) *Let $\mathcal{U} = \{u_0 < \cdots < u_{\|\mu\|-1}\}$ be the indices of the unconstrained bits of a mask μ, such that* $\text{bit}(\mu, u_k) = 1$ *for all* $0 \leq k < \|\mu\|$, *and let π be a pattern with respect to μ. Consider $a \neq b \in \mathcal{I}$. Let i be the index of the most significant bit of a and b that does not match; in other words, $i = \max\{k | \text{bit}(a,k) \neq \text{bit}(b,k)\}$. It follows that $i \in \mathcal{U}$.*

Proof: By Theorem 2.2 we have that

$$\text{bit}(a, i) = \sum_{i \leq k < n} \text{bit}(\text{gc}(a), k) \bmod 2.$$

Knowing that $\text{bit}(a, j) = \text{bit}(b, j)$ for $j > i$, we can use Theorem 2.1 to infer that $\text{bit}(\text{gc}(a), j) = \text{bit}(\text{gc}(b), j)$ for $j > i$. Thus, we have that

$$\begin{aligned} \text{bit}(a,i) + \text{bit}(b,i) &= \sum_{i \leq k < n} \big(\text{bit}(\text{gc}(a),k) + \text{bit}(\text{gc}(b),k)\big) \bmod 2 \\ &= \text{bit}(\text{gc}(a),i) + \text{bit}(\text{gc}(b),i)\,. \end{aligned}$$

Suppose $i \notin \mathcal{U}$. Then it follows that $\operatorname{bit}(\operatorname{gc}(a), i) = \operatorname{bit}(\operatorname{gc}(b), i) = \operatorname{bit}(\pi, i)$, and therefore $\operatorname{bit}(a, i) = \operatorname{bit}(b, i)$, a contradiction. ■

Theorem 3.2 (Gray Code Rank) *Let $\mu, \pi, \mathcal{I}, \mathcal{U}$ and n be as in Lemma 3.1. Consider $i \neq j \in \mathcal{I}$, and define*

$$\bar{i} = \left[\operatorname{bit}\left(i, u_{\|\mu\|-1}\right) \cdots \operatorname{bit}(i, u_0)\right]_{[2]}.$$

Then $i < j$ if and only if $\bar{i} < \bar{j}$. That is, the Gray code rank is given by $\operatorname{gcr}(i) = \bar{i}$.

Proof: Lemma 3.1 tells us that the most significant differing bit between these two values must be in an unconstrained bit position. In other words, the only bits necessary to compare the relative order of i and j are precisely the bits of index $u \in \mathcal{U}$. If we remove the constrained bits from i, and keep the unconstrained bits in the same relative order, we are left with $\bar{i}$. Thus, it follows that $\bar{i}$ and $\bar{j}$ will always have the same relative ordering as i and j. Since $\bar{i}$ is a $\|\mu\|$ digit binary number, it follows by the definition of gcr that $\operatorname{gcr}(i) = \bar{i}$. ■

Algorithm 4 GrayCodeRank(n, μ, π, i)

Given μ, π and n as per Lemma 3.1 and a value $i \in \mathcal{I}$, calculates $r \in \mathbb{B}^{\|\mu\|}$ such that $r = \operatorname{gcr}(i)$.

Input: $n \in \mathbb{Z}_+$, $\mu \in \mathbb{B}^n$ and $i \in \mathcal{I}$.

Output: $r \in \mathbb{B}^{\|\mu\|}$ such that $r = \operatorname{gcr}(i)$.

```
1: $r \leftarrow 0$
2: for $k = n - 1$ to 0 do
3:     if $\operatorname{bit}(\mu, k) = 1$ then
4:         $r \leftarrow (r \lhd 1) \vee \operatorname{bit}(i, k)$
5:     end if
6: end for
```

As per Theorem 3.2, Algorithm 4 computes $\operatorname{gcr}(i)$ given n, μ, π and i. Given $\operatorname{gcr}(i)$ it is natural to want to reconstruct one or both of i and $\operatorname{gc}(i)$. We work in parallel to reconstruct the values of $\operatorname{gc}(i)$ and i given $\operatorname{gcr}(i)$. Since $i \in \mathcal{I}$ it follows that $\operatorname{bit}(\operatorname{gc}(i), k) = \operatorname{bit}(\pi, k)$ for $k \notin \mathcal{U}$. Additionally, when $k \in \mathcal{U}$ it follows that $\operatorname{bit}(i, k) = \operatorname{bit}(\operatorname{gcr}(i), j)$ where $k = u_j$. Given any k, exactly one of $\operatorname{bit}(i, k)$ or $\operatorname{bit}(\operatorname{gc}(i), k)$ is known. Theorem 2.1 lets us fill in the blanks as $\operatorname{bit}(\operatorname{gc}(i), k) = \operatorname{bit}(i, k) + \operatorname{bit}(i, k + 1)$. If we work from the most significant bit to the least significant bit, $\text{bit}(i, k+1)$ will be known at step k, allowing us to solve for the unknown bit. We formalize this procedure in Algorithm 5.

Algorithm 5 GRAYCODERANKINVERSE(n, μ, π, r)

Given μ, π and n as per Lemma 3.1 and a value $r \in \mathbb{B}^{\|\mu\|}$, calculates $i \in \mathcal{I}$, and $\text{gc}(i) \in \mathbb{B}^n$ such that $r = \text{gcr}(i)$.

Input: $n \in \mathbb{Z}_+$, $\mu, \pi \in \mathbb{B}^n$ and $r \in \mathbb{B}^{\|\mu\|}$.
Output: $i \in \mathcal{I}$ such that $r = \text{gcr}(i)$; and $g = \text{gc}(i) \in \mathbb{B}^n$.

```
1: $(i, g, j) \leftarrow (0, 0, \|\mu\| - 1)$
2: for $k$ from $n-1$ to $0$ do
3:     if $\text{bit}(\mu, k) = 1$ then
4:         $\text{bit}(i, k) \leftarrow \text{bit}(r, j)$
5:         $\text{bit}(g, k) \leftarrow \text{bit}(i, k) + \text{bit}(i, k+1) \mod 2$
6:         $j \leftarrow j - 1$
7:     else
8:         $\text{bit}(g, k) \leftarrow \text{bit}(\pi, k)$
9:         $\text{bit}(i, k) \leftarrow \text{bit}(g, k) + \text{bit}(i, k+1) \mod 2$
10:    end if
11: end for
```

3.2 Algorithms

When calculating the Hilbert index, we determine in which side of the half-plane the coordinate $\mathbf{p}$ lies in with respect to each of the axes. The integer l is calculated at each iteration i of the algorithm as

$$\begin{aligned} l &= T_{(e,d)}([\text{bit}(p_{n-1}, i) \cdots \text{bit}(p_0, i)]_{[2]}) \\ &= \big([\text{bit}(p_{n-1}, i) \cdots \text{bit}(p_0, i)]_{[2]} \circlearrowright (d+1)\big) \veebar \big(e \circlearrowright (d+1)\big). \end{aligned}$$

We consider the case where axis j has precision m_j instead of all axes having precision m. Regardless of $\mathbf{p}$ it follows that $\text{bit}(p_j, i) = 0$ when $i \geq m_j$. At iteration i, we define

$$\mu = [\alpha_{n-1} \cdots \alpha_0]_{[2]} \circlearrowright (d+1), \text{ where } \alpha_j = \begin{cases} 1, & \text{if } m_j > i, \\ 0, & \text{otherwise;} \end{cases}$$

and $\pi = \big(e \circlearrowright (d+1)\big) \wedge !\mu$. It can be seen that $l \wedge !\mu = \pi$, thus we may apply Theorem 3.2 to $\text{gc}^{-1}(l)$ to calculate a $\|\mu\|$-bit rank that maintains the same relative ordering as $\mathrm{gc}^{-1}(l)$. Thus, at each iteration i, instead of appending the n-bit value $\mathrm{gc}^{-1}(l)$ to h, we may append the $\|\mu\|$-bit value $\mathrm{gcr}(\mathrm{gc}^{-1}(l))$. Each dimension j will contribute a 1-bit to μ for iterations $0 \leq i < m_j$, each time contributing a single bit to h. Thus, each dimension j will contribute *exactly* m_j bits to h, yielding a final index $M = \sum_j m_j$ bits in length. As desired, the constructed compact Hilbert code will have the same precision as the original point $\mathbf{p}$. We formalize this approach with Algorithms 6 and 7. The inverse procedure is equally straight-forward and is shown in Algorithm 8.

Given the tools presented in this paper, it is relatively straight-forward to construct algorithms for efficiently iterating through all points on a regular or compact Hilbert curve, as well as for calculating various other quantities as per Moore [22].

4 Conclusion

Due their wide variety of uses and simple elegance, space-filling curves have been researched continuously since their discovery over a century ago. Various types of curves have been created and many of them have found application in real-world problems. Because of this, researchers have been driven to create algorithms for the efficient traversal and indexing of these curves. Motivated by the lack of clarity and intuition in the near-ubiquitous Butz [8] algorithms as implemented by Moore [22], we have reconstructed these algorithms from a natural geometric point of view, with detail and rigor.

Furthermore, we identified an inefficiency in the use of space-filling curves in database systems for real-world data-sets and identified a solution. We discussed the concept of Hilbert curves over spaces with differently sized dimensions, and introduced the idea of compact Hilbert indices.

Finally, we developed algorithms for dealing with compact Hilbert indices. While yielding a family of algorithms nearly identical in implementation to those of Moore, it is hoped that the path travelled to arrive at them has been intuitive and illuminating. Finally, it is hoped that the introduced notion of compact Hilbert indices will prove fruitful and find application in database systems.

Algorithm 6 ExtractMask$(n, m_0, \ldots, m_{n-1}, i)$

Extracts a mask μ indicating which axes are active at a given iteration i of the CompactHilbertIndex algorithm.

Input: $n, m_0, \ldots, m_{n-1} \in \mathbb{Z}_+$ and $i \in \mathbb{Z}_n$.

Output: The mask μ of active dimensions at iteration i.

```
1: $\mu \leftarrow 0$
2: for $j = n-1$ to $0$ do
3:     $\mu \leftarrow \mu \lhd 1$
4:     if $m_j > i$ then
5:         $\mu \leftarrow \mu \vee 1$
6:     end if
7: end for
```

Algorithm 7 CompactHilbertIndex$(n, m_0, \ldots, m_{n-1}, \mathbf{p})$

Calculates the compact Hilbert index $h \in \mathbb{B}^M$ of a point $\mathbf{p} \in \mathbb{P}$.

Input: $n, m_0, \ldots, m_{n-1} \in \mathbb{Z}_+$ and a point $\mathbf{p} \in \mathbb{P}$.

Output: $h \in \mathbb{B}^H$, the compact Hilbert index of the point $\mathbf{p} \in \mathbb{P}$.

```
1: $(h, e, d) \leftarrow (0, 0, 0)$
2: $m \leftarrow \max_i\{m_i\}$
3: for $i = m-1$ to $0$ do
4:     $\mu \leftarrow$ ExtractMask$(n, m_0, \ldots, m_{n-1}, i)$
5:     $\mu \leftarrow \mu \circlearrowright (d+1)$
6:     $\pi \leftarrow (e \circlearrowright (d+1)) \wedge !\mu$
7:     $l \leftarrow [\mathrm{bit}\,(p_{n-1}, i) \cdots \mathrm{bit}\,(p_0, i)]_{[2]}$
8:     $l \leftarrow T_{(e,d)}(l)$
9:     $w = \mathrm{gc}^{-1}(l)$
10:    $r =$ GrayCodeRank$(n, \mu, \pi, w)$
11:    $e \leftarrow e \veebar (\mathrm{e}(w) \circlearrowleft (d+1))$
12:    $d \leftarrow d + \mathrm{d}(w) + 1 \bmod n$
13:    $h \leftarrow (h \lhd \|\mu\|) \vee r$
14: end for
```

Algorithm 8 CompactHilbertIndexInverse$(n, m_0, \ldots, m_{n-1}, h)$

Calculates the point $\mathbf{p} \in \mathbb{P}$ corresponding to a given compact Hilbert index $h \in \mathbb{B}^M$.

Input: $n, m_0, \ldots, m_{n-1} \in \mathbb{Z}_+$ and $h \in \mathbb{B}^M$, the compact Hilbert index of the point $\mathbf{p}$.

Output: A point $\mathbf{p} \in \mathbb{P}$.

```
 1: $(e, d, k) \leftarrow (0, 0, 0)$
 2: $\mathbf{p} = [p_0, \ldots, p_{n-1}] \leftarrow [0, \ldots, 0]$
 3: $m \leftarrow \max_i\{m_i\}$
 4: $M \leftarrow \sum_i\{m_i\}$
 5: for $i = m - 1$ to $0$ do
 6:     $\mu \leftarrow \textsc{ExtractMask}(n, m_0, \ldots, m_{n-1}, i)$
 7:     $\mu \leftarrow \mu \circlearrowright (d + 1)$
 8:     $\pi \leftarrow (e \circlearrowright (d + 1)) \wedge !\mu$
 9:     $r \leftarrow [\mathrm{bit}\,(h, M - k - 1) \cdots \mathrm{bit}\,(j, M - k - \|\mu\|)]_{[2]}$
10:     $k \leftarrow k + \|\mu\|$
11:     $w \leftarrow \textsc{GrayCodeRankInverse}(n, \mu, \pi, r)$
12:     $l = \mathrm{gc}(w)$
13:     $l \leftarrow T^{-1}_{(e,d)}(l)$
14:     for $j = 0$ to $n - 1$ do
15:         $\mathrm{bit}\,(p_j, i) \leftarrow \mathrm{bit}\,(l, j)$
16:     end for
17:     $e \leftarrow e \veebar (\mathrm{e}(w) \circlearrowleft (d + 1))$
18:     $d \leftarrow d + \mathrm{d}(w) + 1 \bmod n$
19: end for
```

References

[1] D. J. Abel and D. M. Mark. A comparative analysis of some two-dimensional orderings. *International Journal of Geographic Information Systems*, 4(1):21–31, January 1990.

[2] C. Alpert and A. Kahng. Multi-way partitioning via spacefilling curves and dynamic programming. In *Proceedings of the 31st Annual Conference on Design Automation*, pages 652–657, San Diego, California, June 6-10 1994.

[3] J. J. Bartholdi III. A routing system based on spacefilling curves. `http://www2.isye.gatech.edu/~jjb/mow/mow.pdf`, April 1995.

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

[5] C. Böhm, S. Berchtold, and D. A. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Computing Surveys*, 33(3):322–373, September 2001.

[6] G. Breinholt and C. Schierz. Algorithm 781: Generating Hilberts space-filling curve by recursion. *ACM Transactions on Mathematical Software*, 24(2):184–189, June 1998.

[7] A. R. Butz. Convergence with Hilbert's space-filling curve. *Journal of Computer and System Sciences*, 3(2):128–146, May 1969.

[8] A. R. Butz. Alternative algorithm for Hilbert's space-filling curve. *IEEE Transactions on Computers*, pages 424–426, April 1971.

[9] S. Chatterjee, A. R. Lebeck, P. K. Patnala, and M. Thottethodi. Recursive array layouts and fast parallel matrix multiplication. In *Proceedings of the Eleventh Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA 1999*, pages 222–231, Saint-Malo, France, June 27-30 1999.

[10] F. Gray. Pulse code communication. US Patent Number 2,632,058, March 17 1953.

[11] D. Hilbert. Über die stetige Abbildung einer Linie auf ein Flächenstück. *Mathematische Annalen*, 38:459–460, 1891.

[12] G. Jin and J. M. Mellor-Crummey. SFCGen: A framework for efficient generation of multi-dimensional space-filling curves by recursion. *ACM Transactions on Mathematical Software*, 31(1):120–148, March 2005.

[13] G. Jin, J. M. Mellor-Crummey, and R. J. Fowler. Increasing temporal locality with skewing and recursive blocking. In *Proceedings of the 2001 ACM/IEEE Conference on Supercomputing*, page 43, Denver, Colorado, November 10-16 2001.

[14] M. Kaddoura, C.-W. Ou, and S. Ranka. Partitioning unstructured computational graphs for nonuniform and adaptive environments. *IEEE Parallel and Distributed Technology: Systems and Technology*, 3(3):63–69, September 1995.

[15] I. Kamel and C. Faloutsos. Hilbert R-tree: An improved R-tree using fractals. In *Proceedings of the Twentieth International Conference on Very Large Databases*, pages 500–509, Santiago, Chile, September 1994.

[16] C.-H. Lamaque and F. Robert. Image analysis using space-filling curves and 1d wavelet bases. *Pattern Recognition*, 29(8):1309–1322, August 1996.

[17] J. K. Lawder. Calculations of mappings between one and n-dimensional values using the Hilbert space-filling curve. Technical Report JL1/00, Birkbeck College, University of London, August 2000.

[18] J. K. Lawder and P. J. H. King. Querying multi-dimensional data indexed using the Hilbert space-filling curve. *SIGMOD Record*, 30(1):19–24, March 2001.

[19] Y. Matia and A. Shamir. A video scrambling technique based on space filling curves. In *Proceedings of Advances in Cryptology - CRYPTO'87*, pages 398–417, Santa Barbara, California, August 16-20 1987.

[20] B. Moghaddam, K. Hintz, and C. Stewart. Space-filling curves for image compression. In *Proceedings of the First Annual SPIE Conference on Automatic Object Recognition*, volume 1471, pages 414–421, Orlando, Florida, April 1-5 1991.

[21] B. Moon, H. Jagadish, C. Faloutsos, and J. H. Saltz. Analysis of the clustering properties of the Hilbert space-filling curve. *Knowledge and Data Engineering*, 13(1):124–141, January 2001.

[22] D. Moore. Fast hilbert curve generation, sorting, and range queries. Internet: `http://web.archive.org/web/20050212162158/http://www.caam.rice.edu/~dougm/twiddle/Hilbert/`, 1999.

[23] E. A. Patrick, D. R. Anderson, and F. Bechtel. Mapping multidimensional space to one dimension for computer output display. *IEEE Transactions on Computers*, 17(10):949–953, October 1968.

[24] G. Peano. Sur une courbe, qui remplit toute une aire plane. *Mathematische Annalen*, 36:157–160, 1890.

[25] L. K. Platzman and J. J. Bartholdi III. Spacefilling curves and the planar travelling salesman problem. *Journal of the ACM*, 36(4):719–737, October 1989.

[26] S. W. Thomas. Utah raster toolkit. Internet: `http://web.mit.edu/afs/athena/contrib/urt/src/urt3.1/urt-3.1b.tar.gz`, 1991.

[27] Y. Zhang and R. E. Webber. Space diffusion: An improved parallel halftoning technique using space-filling curves. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 93*, pages 305–312, Anaheim, California, August 2-6 1993.

A Notation

In the following table, we summarize and briefly explain the notation used throughout this document.

Notation	
$[\cdot]_{[2]}$	Used to denote non-negative integers written in base 2.
$\mathbf{x}$	Bold-faced font is used to represent vectors.
$\mathrm{bit}\,(a,k)$	Represents the value of the kth bit of a non-negative integer a.
Operators	
$\lVert\cdot\rVert$	Number of '1' bits in the binary representation of a non-negative integer (the parity).
$\lvert\cdot\rvert$	The cardinality of a set.
$\vee$	The bitwise *or* operator.
$\veebar$	The bitwise *exclusive-or* operator.
$\wedge$	The bitwise *and* operator.
$\bar{\wedge}$	The bitwise *not-and* operator.
!	The bitwise *not* operator.
$\triangleleft$	The bitwise *shift-left* operator.
$\triangleright$	The bitwise *shift-right* operator.
$\circlearrowleft$	The bitwise *left-rotation* operator.
$\circlearrowright$	The bitwise *right-rotation* operator.
Functions	
tsb	The *trailing set bits* function.
gc	The *binary reflected Gray code* function.
gcr	The *Gray code rank* function.
Spaces	
$\mathbb{Z}$	The set of integers, $\{\ldots,-1,0,1,\ldots\}$.
$\mathbb{Z}_+$	The set of positive integers, $\{1,2,\ldots\}$.
$\mathbb{N}$	The set of natural integers, $\{0,1,\ldots\}$.

Continued on next page...

...continued from previous page.	
$\mathbb{Z}_k$	The set of integers modulo k, $\{0, \ldots, k-1\}$.
$\mathbb{B}$	The set of integers $\{0, 1\}$.
$\mathbb{B}^k$	The set of positive integers of k bits, $\mathbb{Z}_{2^k}$.
$\mathbb{P}$	The n-dimensional space $\mathbb{B}^{m_0} \times \cdots \times \mathbb{B}^{m_{n-1}}$.
Sequences	
$m_0, \ldots, m_{n-1}$	Precision (number of bits) of each of the n dimensions.
$\mathrm{g}(0), \ldots, \mathrm{g}(2^n - 2)$	Sequence of integers in $\mathbb{Z}_n$ such that $\mathrm{gc}(i) \veebar 2^{\mathrm{g}(i)} = \mathrm{gc}(i+1)$.
$\mathrm{e}(0), \ldots, \mathrm{e}(2^n - 1)$	Sequence of *entry points* in $\mathbb{B}^n$.
$\mathrm{f}(0), \ldots, \mathrm{f}(2^n - 1)$	Sequence of *exit points* in $\mathbb{B}^n$.
$\mathrm{d}(0), \ldots, \mathrm{d}(2^n - 1)$	Sequence of *directions* in $\mathbb{Z}_n$ such that $\mathrm{e}(i) \veebar 2^{\mathrm{g}(i)} \veebar 2^{\mathrm{d}(i)} = \mathrm{e}(i+1)$.
Values	
$\mathbf{p} = [p_0, \ldots, p_{n-1}]$	A point in the space $\mathbb{P}$.
n	Dimensionality of the space $\mathbb{P}$.
m	Maximum precision, $m = \max_i\{m_i\}$.
e	An *entry point* in $\mathbb{B}^n$.
f	An *exit point* in $\mathbb{B}^n$.
d	A *direction* in $\mathbb{Z}_n$.
M	The net precision, $M = \sum_i m_i$.
h	A Hilbert index in $\mathbb{B}^M$.
μ	A *mask* in $\mathbb{B}^n$.
π	A *pattern* in $\mathbb{B}^n$ such that $\pi \wedge \mu = 0$.
Sets	
$\mathcal{I}$	The set of points $\{i \in \mathbb{B}^n \mid \mathrm{gc}(i) \wedge !\mu = \pi\}$.
$\mathcal{U} = \{u_0 < \ldots < u_{\|\mu\|-1}\}$	The set of unconstrained bits associated with a given mask μ.