Computer Reviews Journal Vol 3 (2019) ISSN: 2581-6640 http://purkh.com/index.php/tocomp

Computation of Lebesgue's Space-Filling Curve

Arthur R. Butz

Department of Electrical and Computer Engineering, Northwestern University

2145 Sheridan Road, Evanston, IL 60208-3118

butz@ece.northwestern.edu

Abstract

The means of realizing or approximating the Lebesgue space-filling curve (SFC) with binary arithmetic on a uniformly spaced binary grid are not obvious, one problem being its formulation in terms of ternary representations; that impediment can be overcome via use of a binary-oriented Cantor set. A second impediment, namely the Devil's Staircase feature, also created by the role of the Cantor set, can be overcome via the definition of a "working inverse," thereby providing means of achieving compatibility with such a grid. The results indicate an alternative way to proceed, in realizing an approximation to Lebesgue's SFC, which circumvents any complication raised by Cantor sets and is compatible with binary and integer arithmetic. Well-known constructions such as the z-curve or Morton order, sometimes considered in association with Lebesgue's SFC, are treated as irrelevant.

Introduction

The unit interval $R = [0,1]$ and the unit square $\boldsymbol{X} = \{R,R\}$. A SFC is a continuous mapping from R onto $\boldsymbol{X}$.

The first space-filling curve (SFC) was proposed by Peano [1], whose construction is based on ternary representations as distinct from the later binary-based Hilbert SFC [2], Such curves are continuous everywhere and not differentiable, and are referred to here as being of the "Peano class"; the Hilbert class is a sub-class of the Peano class. No SFC is 1-to-1, i.e., possessing an inverse. Quantized versions of the Peano and Hilbert curves have been generalized to n-dimensional spaces [3-6].

SFCs have been used or proposed in diverse applications, for example numerical computations [7-10], image processing and retrieval [11-12], computational fluid dynamics [13], data storage [14-20], antenna design [21], reservoir simulation [22], pattern analysis [23], geophysical data processing [24], VLSI architectures [25], scheduling in parallel processing [26], graph layout [27], and MR imaging [28]. The Hilbert SFC has been favored in applications partly because it is naturally compatible with binary representations and a binary-based rectangular grid in the n-dimensional space of interest; a quantized version of the Hilbert SFC and its inverse, for general dimension n, have been coded by Moore [29]. No SFC in the Peano class is differentiable, which can be a drawback, especially in numerical applications.

Lebesgue's SFC has been described by Sagan [30] and is evidently disfavored in applications of SFCs, one reason being perhaps its poor "compaction" properties, which is the tendency of the curve to remain in some small neighborhood of space once that neighborhood has been entered in filling the space. Another discouraging feature, from computation, has been its expression in terms of ternary representations via the standard or generic Cantor set. A third discouraging feature has been the lack, filled here, of a specific way to generate a quantized version. Finally, Lebesgue's SFC has an important "Devil's Staircase" feature that might seem meaningless on a finite grid. Lebesgue's SFC is differentiable a.e., a feature that can be desirable in some applications. A credible and useful realization of Lebesgue's SFC on a finite binary grid is the central result here. The "working inverse" of Lebesgue's SFC is the key to achieving that. Our result is unrelated to the "Morton order" or "z-curve", though the Lebesgue SFC is sometimes viewed as a version of, or helper to, the Morton order. Haverkort [31] and Bader [14] noted the Morton order is not an SFC at all, i.e., its expression in a continuum

is not continuous. Both Haverkort and Bader, with general remarks, treated Lebesgue's SFC as a fix for the discontinuous Morton order, but the details are not clear and important possibilities, such as an inverse, which naturally should apply to realizations on a finite binary grid, are not confronted. We treat the Morton order and z-curve as irrelevant and derive results and conclusions from Lebesgue's SFC as proposed, but using an alternative Cantor set.

Lebesgue's description of his SFC was terse [32]. Earlier SFCs had not been differentiable, but Lebesgue's is differentiable a.e..

The principal objective here is to express Lebesgue's SFC in a way compatible with binary arithmetic, bitwise operations on integer representations, and a binary grid. An alternative Cantor set is used in formulating, for the case dimension $n = 2$, a SFC which follows Lebesgue's but averts practical complications arising from ternary representations. The result is the same SFC, i.e. tracing the same path through $\boldsymbol{X}$, with a different relationship to its index or key $r \in R$. Despite that difference, we continue to refer to the focus of our study and our result as Lebesgue's SFC, since the path is the same. The new mapping remains differentiable a.e., and naturally compatible with binary and integer representations of variables, easing the tasks of computations associated with it. However, Lebesgue's SFC does not naturally "fill" a rectangular grid consisting of a finite number of uniformly binary-spaced points; that arrangement requires substantial supplementation. That is done here via the choice of a "working inverse" defined via a recursive computation. Code is provided in Appendix B for computing this SFC and the working inverse on a binary grid. The code presented here uses real rather than integer representations for the sake of clarity and transparency only; integer representations and computations, involving shifts replacing multiplications and divisions, could obscure matters, but code using integer representations of the crucial variables is available as of Jan. 2020 here.

The compatibility with a binary-spaced rectangular grid raises the candidacy of our result in some applications where the Hilbert class has been used and even suggests that Hilbert-Lebesgue hybrid SFCs can be considered, having both compaction and differentiability properties.

Materials and Methods

There were no special materials required for this research; commonplace computational resources are all that is required to reproduce these results following the methods described here.

The Cantor Set ⬩

Number representations are presumed to be of the conventional decimal type unless noted otherwise, e.g. $0_2.01 = 0_2.00111111111\ldots\ldots = 0_3.02020202\ldots\ldots = 0_8.2 = 0_9.2222\ldots$ are binary, ternary, octal and nonal (base 9) representations, respectively, of $r = 0.25 = ¼$.

The generic Cantor set ⬩, commonly defined via the usual ternary scheme, may also be defined in terms of nonal (base 9) representation allowing only the characters 0,2,6,8; such representations are of those values which, in ternary representation, use no 1s, as required for the generic Cantor set. Octal representations of $r \in R$ that allow only the values 0,2,5,7 can be associated with nonal representations that use only 0,2,6,8 by defining the correspondences [0,2,6,8] ↔ [0,2,5,7] respectively.

In this way a Cantor set ⬩⬩⬩ perfect, nowhere dense and of measure 0 is defined, and it is suggested that a new SFC, resembling Lebesgue's, can be constructed. For example the critical points 1/9,2/9,1/3,2/3,7/9,8/9 ∈ ⬩ ($0_9.08888\ldots$, $0_9.2$, $0_9.28888\ldots$, $0_9.6$, $0_9.68888\ldots$, $0_9.8$) correspond via the homeomorphism, respectively, to the points ⅛,¼, ⅜, ⅝, ¾, ⅞ ($0_8.0777777\ldots$, $0_8.2$, $0_8.277777\ldots$, $0_8.5$, $0_8.57777\ldots$, $0_8.7$). This alternative Cantor set is not equivalent to the Smith-Volterra-Cantor set, which it may resemble; the Smith-Volterra-Cantor set has measure ½ while ⬩, like the generic Cantor set ⬩, has measure 0. To show that, starting with (0,1) do an interval removal process but starting with removal of the three intervals (⅛,¼)∪(⅜,⅝)∪(¾,⅞), whose total measure is

½, as constituting one step. Remaining are four intervals of measure ⅛ each;. if the first removal process on each of these reduced intervals is done, the measure of those removed on this second step is ⅛·½·4 = ¼. The total measure of all intervals removed is thus ½ + ¼ + ⅛ + ... = 1.

The main motivation for the definition of ・・ is to escape awkward ternary-related computations, thereby making feasible writing code in terms of integer types, and making possible needed multiplications and divisions with shifts and adds, and to arrange that vertical and horizontal sensitivities of the SFC to be defined are equal on any one "level" (a concept to be defined). The conclusions reached, and suggestions for further work formulated here, could cast a different light on such objectives.

Formulation

This is the procedure for defining the SFC. For a point r in ・, define $\boldsymbol{x}$ = ・(r) as Lebesgue did for the corresponding $r \in$ ・・, which assured ・ is continuous on ・; the details are given e.g. by Sagan [30]. For an interval bounded by points in ・, but with no interior points in ・, such as [⅜,⅝], perform the linear interpolations between endpoints equal to those used for the corresponding interval e.g. [1/3,2/3], for the Lebesgue SFC.

The procedure creates a SFC differentiable a.e., here called ・:$R\rightarrow$onto$\boldsymbol{X}$, which "fills" $\boldsymbol{X}$ in two senses. As the zero-measure generic Cantor set ・ has the same cardinality as the unit interval R and the unit square $\boldsymbol{X}$, the image of ・ as defined is the entire square $\boldsymbol{X}$, so that much fills $\boldsymbol{X}$ but is not a mapping from R, i.e. it is not a "curve".

Lebesgue's ・(r), and our ・(r), for $r \notin$ ・・ or・ ・ respectfully, provide a second filling, or more specifically a differentiable mapping whose closure fills $\boldsymbol{X}$, as that second filling naturally does via the linear interpolations. The second filling is more compatible with computation on a binary-spaced grid, but not entirely so for ・, since computations of ・ remain naturally compatible with ternary rather than binary arithmetic. The Cantor set ・, with its associated ・・(to be defined)・・・eliminates any relevance of ternary-based computations. The definition of a "working inverse," proposed here, is however crucial to our method of filling a rectangular binary-spaced grid in a quasi-continuous way.

Definitions

A mapping ・ : $R\rightarrow X$, which is to be used to define the SFC, takes ・(0) = {0,0} and ・(1) = {1,1}. The set ・$_0 \subset R$ is defined as [⅛, ¼] ∪ [⅜, ⅝] ∪ [¾, ⅞]. On ・$_0$ define:

If ⅛ ≤ r ≤ ¼ then x_0 = 1-4r and x_1 = ½, i.e. ・(r) = {1-4r, ½},

If ⅜ ≤ r ≤ ⅝ then ・(r) = {½, 2.5 – 4r}, (1)

If ¾ ≤ r ≤ ⅞ then ・(r) = {4-4r, ½},

The relations (1) are the linear interpolations, chosen by Lebesgue, between endpoints in ・ also chosen by him; our endpoints are those in ・ that equal his for corresponding intervals.

The trajectories in Fig. 1 (Level 0 case) show the subset of $\boldsymbol{X}$ that the definition (1) encompasses. On ・$_0$ either d・$_0/dr$ = 0 and d・$_1/dr$ = -4 or vice versa, i.e. the vertical and horizontal sensitivities on the points of the unit square X, that are depicted in Fig. 1, are equal. The set B_0 is analogous to Lebesgue's set [1/9, 2/9] ∪ [1/3, 2/3] ∪ [7/9, 8/9], for which a function comparable to our ・, namely ・, is defined but the vertical and horizontal sensitivities differ for ・

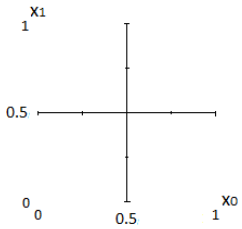

Fig. 1. Level 0 (L0) case.

Complete the definition of ・:

If 0 < r < ⅛ then ・(*r*) = ・(8r)/2.

If ¼< *r* < ⅜ then ・(*r*) = ・(8r-2)/2 + {0, ½}. (2)

If ⅝ < *r* < ¾ then ・(*r*) = ・(8r-5)/2 + {½,0}.

If ⅞ < *r* < 1 then ・(*r*) = ・(8r-7)/2 + {½,½}.

The extension (2) to (1) must be interpreted in a recursive sense. According to the given definition ・(1/128) = ・(1/16)/2 = ・(½)/4 = {⅛,⅛}. If *r* ∈ ・ is not a member of the countable subset of ・ serving as endpoints of intervals, or within one of those intervals, then the recursion does not terminate. *r* = 2/7 = 0_8.222222……. is a member of that uncountable subset and if we wish *y* = ・(*r*) then applying the preceding recursion *y* = *y*/2 + {0,0.5} = *y*/4 + {0,0.75} = *y*/8 + {0,0.875} = …… converging to {0,1}. Thus (1-2) define ・ only a.e.; that constitutes no difficulty here, because the objective is a quantized version of the Lebesgue SFC.

The preceding is, mutatis mutandis, Sagan's description [30] of the part of the Lebesgue SFC defined on intervals bounded by some members of ・ ・ ・. The trajectories in ***X*** defined by Lebesgue are the same as those defined here; the difference is in the relationship to the index *r*. The definition of ・ in (1) represents linear interpolations between endpoints on ・. that correspond to those used on corresponding endpoints in ・ used to define linear interpolations for Lebesgue's SFC for index *r* values ∉ ・. Therefore, that part of the SFC is the same as Lebesgue's, though the relationship to the index *r* is changed. Consequences of the new formulation are that computations of ・(*r*) may dodge some computational problems raised by ternary representations and vertical and horizontal sensitivities, on any one level, are equal.

It is possible to make the following observations from (1) and (2); where {x_0,x_1} = ・(*r*):

Obs. 1: 0 ≤ *r* ≤ ⅛ ⇒ x_0 ≤ ½ and x_1 ≤ ½. [0, ⅛] maps into the lower left quadrant of ***X***

Obs. 2: ¼ ≤ *r* ≤ ⅜ ⇒ x_0 ≤ ½ and ½ ≤ x_1 ≤ 1. [⅛, ¼] maps into the upper left

Obs. 3 ⅝ ≤ *r* ≤ ¾ ⇒ ½ ≤ x_0 ≤ 1 and x_1 ≤ ½. [⅝, ¾] maps into the lower right

Obs. 4: ⅞ ≤ *r* ≤ 1 ⇒ ½ ≤ x_0 ≤ 1 and ½ ≤ x_1 ≤ 1 . [⅞, 1] maps into the upper right

The following observations are not quite converses of Observations 1-4:

Obs. 5: x_0 < ½ and x_1 < ½ ⇒ 0 ≤ *r* < ⅛ .

Obs. 6. x_0 < ½ and x_1 > ½ ⇒ ¼ < *r* < ⅜,

Obs. 7. x_0 > ½ and x_1 < ½ ⇒ ⅝ < *r* < ¾ ,

Obs. 8. $x_0 > ½$ and $x_1 > ½ \Rightarrow ⅞ < r \leq 1$.

The validity of Obs. 5 is seen by excluding the 7 other intervals, and Obs. 6,7&8 follow similarly.

Description of the Curve.

The definition (2) of ・ defines paths in the four quadrants of ***X*** depicted in Fig. 2, referred to as Level 1 cases.

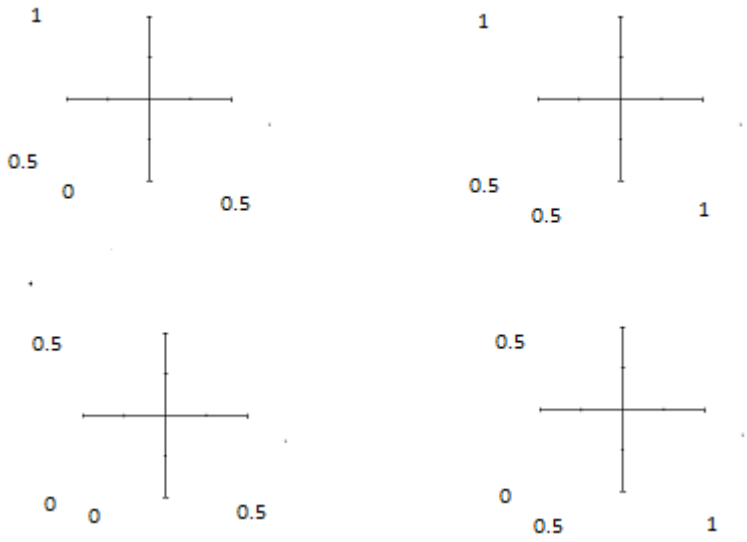

Fig. 2. Four Level 1 (L1) cases

As for the concept "level," henceforth the notation L*k* is used to refer to the Level *k* case. There are 4^k L*k* cases; each represents crossed trajectories of length 2^{-k} (Fig. 2 shows the four L1 cases). Borders of squares are not part of any level, except for endpoints of crossed axes. Actually, it is more convenient to view the horizontal trajectories, in each case, as two trajectories joined at the middle of the vertical trajectory.

The curve ***x*** = ・(*r*) on examination, and noting Obs. 1-8, therefore is described as follows. As *r*, starting from 0, progresses monotonically toward 1, ***x*** = ・(*r*) starts at {0,0} and progresses somewhat circuitously toward ***x*** = {½,½}, remaining within the bounds $0 \leq x[0] \leq ½$, $0 \leq x[1] \leq ½$, but attaining ***x*** = {½,½} only when *r* = ⅛. Then *r*, starting from ⅛ and with ***x*** starting at {½,½}, progresses to ¼ where ***x*** = {0, ½} and then to *r* = ⅜ where ***x*** = {½,1}, remaining within the bounds $0 \leq x[0] \leq ½$, $½ \leq x[1] \leq 1$, attaining ***x*** = {½,1} only when *r* = ⅜. Then as *r* increases monotonically toward, and attaining, *r* = ⅝, only x_1 varies as ***x*** progresses toward and attains {½,0}. ***x*** = ・(*r*) proceeds from {½,0} to progresses circuitously toward ***x*** = {1,0}, remaining within the bounds $½ \leq x[0] \leq 1$, $0 \leq x[1] \leq ½$, but attaining ***x*** = {1,½} only when *r* = ¾. Then *r*, starting from ¾ and with ***x*** starting at {1,½}, progresses to ⅞ where ***x*** = {½, ½} and then to *r* = 1 where ***x*** = {1,1}, remaining within the bounds $½ \leq x[0] \leq 1$, $½ \leq x[1] \leq 1$, attaining ***x*** = {1,1} only when *r* = 1.

The process of recursive definition can be carried on indefinitely, but in order to produce a finite binary-spaced grid it must terminate at some point. That raises problems that render the curve, as defined to this point, not a quantized SFC; it must be somehow supplemented. The problem is in dealing with a "Devil's Staircase" problem in the quantized case.

According to (1) and (2) and the fact of recursion in computing ・(*r*), neither x_0 nor x_1 can increase as *r* increases, within one level, in the computation of $\{x_0, \underline{x}_1\}$ = ・(*r*); indeed exactly one of the two coordinates decreases. The problem, then, is how does the image of *r*, i.e. ・(*r*), pass continuously from {0,0} toward {1,1}? This is the Devil's Staircase (discussed by Sagan [30]) feature of Lebesgue's SFC. The answer provided by Lebesgue's formulation

to the question posed is that continuous vertical and horizontal paths toward {1,1} are provided by the applicable Cantor set, which is uncountable but our formulation has this far employed only a countable subset. Since there is no such thing as a quantized Cantor set then provision of a path toward {1,1} is a problem.

The need can be satisfied because the points $\boldsymbol{x}$ in the rectangular grid $\cdot$ must map onto are predetermined. What is needed is some sort of inverse of $\cdot$.

Working Inverse

Let N be a power of 2 and $\boldsymbol{X}(N)$ a binary grid consisting of 2^{2N} values $\{k_0 2^{-N}, k_1 2^{-N}\}$, $k_i = 0, 1, 2, \ldots, 2^N-1$ in $\boldsymbol{X}$. The "working inverse" is designated $\cdot : \boldsymbol{X}(N) \rightarrow R$; with $\cdot(\{0,0\}) = 0$. For the points in Fig. 1 (L0 case) the unique working inverse ω is defined by reversing the relations in (1) as follows:

if $x_0 = ½$ then $r = \cdot(\boldsymbol{x}) = (2.5\text{-}x_1)/4$; return this value

else if $x_1 = ½$ (3)

if $x_0 < ½$ then $r = \cdot(x) = (1\text{-}x_0)/4$; return this value

else if $x_0 > ½$ then $r = \cdot(x) = 1 - x_0/4$; return this value

This choice and the recursive computational algorithm is the basis for the "working inverse"; the alternative candidate inverses of $\boldsymbol{x} = \{½,½\}$ are excluded by the design of the code; $\boldsymbol{x}_0 = ½$ is tested before $\boldsymbol{x}_1 = ½$ is tested.

If an inverse is not found by the test (3) at L0, the next higher level is tried, as clarified in Appendix B. Computation of the working inverse on a binary grid is achieved by regarding coordinates as translated and reduced versions of the L0 case, for which a unique working inverse has been defined for each point. The computation then proceeds recursively by finding the lowest level (minimum k as in Lk) for which the index r can be computed, so the working inverse $\cdot(\boldsymbol{x})$ can be computed only when $\boldsymbol{x}$ is on a reduced (by a power of 2 in both directions) and translated version of the lines in Fig.1. With that understanding, (3) together with the recursive algorithm define the working inverse $\cdot$, but only on a finite binary-spaced grid.

The present development presumes for definition of $\cdot(\boldsymbol{x})$ that at least one coordinate of $\boldsymbol{x}$ is on a finite binary-space grid corresponding to the crossed axes of the various levels. For example, the point $\boldsymbol{x} = \{0.5, \cdot/4\}$ is not on a binary-spaced grid of any interest, but it is encompassed by the L0 case, so according to (3) $\cdot(\boldsymbol{x}) = (2.5 - \cdot/4)/4$. Higher levels are computed by recursion. The L1 cases, for example, are computed by applying to the intervals (0, ⅛), (¼,⅜), (⅝,¾) and (⅞, 1) respectively, by recursion of the algorithm, the measures $\cdot$ applied to L0.

As another example, $\cdot\cdot\{1/4,1/2\}) = 3/16$ by the given definition, but the recursive algorithm yields $\cdot(3/64) = \cdot(21/64) = \{1/4,1/2\}$ as well. The resolution of the ambiguity is in two observations:

1. It is only required, for present purposes, that $\cdot\cdot\omega(\boldsymbol{x}))$ is an identity operation, not $\omega\cdot\cdot(r))$.

2. The algorithm provided here for a "working inverse" is recursive and of a nature that resolves any such ambiguities; every point in the L0 case has a unique working inverse as defined and all computations, of $\cdot\cdot$ and $\cdot\omega$, are recursive and reference the L0 case as foundation or base case. In order to encounter an ambiguity, a higher level would have to be encountered, but that higher level is not encountered because the recursive algorithm has returned a result and been exited.

In this construction the working inverse is defined only if one of the two components of X is binary rational and both are in [0,1]. The definition provides $\cdot(2/7) = \{0,1\}$ and $\cdot(5/7) = \{1,0\}$, as indicated above, but those points are not on the binary grid defined, i.e., they are not included in any level. An attempt to compute $\cdot(2/7)$ or $\cdot(5/7)$ via the recursive algorithms results in infinite recursion.

Definition. Four sub-squares of $\boldsymbol{X}$ are defined by

$\boldsymbol{X}_{01} : 0 \le x_0 < ½, ½ < x_1 \le 1$ $\boldsymbol{X}_{11} : ½ < x_0 \le 1, ½ < x_1 \le 1$

$\boldsymbol{X}_{00} : 0 \le x_0 < ½, 0 \le x_1 < ½$ $\boldsymbol{X}_{10} : ½ < x_0 \le 1, 0 \le x_1 < ½$

Theorem 1. Assuming each x_{ij} is in some $\boldsymbol{X}(N)$, $\cdot(\boldsymbol{X}_{00}) < ⅛ < ¼ < \cdot(\boldsymbol{X}_{01}) < ⅜ < ⅝ < \cdot(\boldsymbol{X}_{10}) < ¾ < ⅞ < \cdot(\boldsymbol{X}_{11}) <$

The proof follows directly from Obs. 5-8.

From this point on we may use, without danger of confusion and when a value of N is understood, notation such as $\cdot(\{k_0,k_1\})$ as short for $\cdot(\{k_0 2^{-N}, k_1 2^{-N}\})$.

Computation of $\cdot$ and $\cdot$.

The strategy defines the "working inverse" to reveal what points in $\boldsymbol{X}(N)$ would be reached by $\cdot(r)$, on a binary grid, after passing through segments of a Cantor set. A recursive construction of the SFC and the working inverse on a finite grid of uniformly-spaced binary rational points can proceed according to the code in Appendix B. The procedure is to compute working inverses (values of r) on the binary grid $\boldsymbol{X}(N)$ using the function omega, then use function beta to compute values of $\cdot(r)$ in $\boldsymbol{X}$ to be checked against the starting grid values. All results can be written to a file, in which the values of r are probably not in natural ascending order. If natural order is needed a sorting program (also provided in Appendix B) can be applied to the results in order to generate a file in which the values of the index r are given in ascending order, with corresponding values of $\boldsymbol{x}$. The computations reveal an SFC applicable only to a binary-spaced grid. A strategy to eliminate the need to sort is indicated below.

Figs 3,4,5 show the development of the computed curve, for $\boldsymbol{X}$ quantized so that $x_i = k_i 2^{-N}$, $k_i = 0,1,2, \ldots, 2^N-1$, for the cases $N = 3,4,5$ respectively. The axes indicate values of k_0 and k_1.

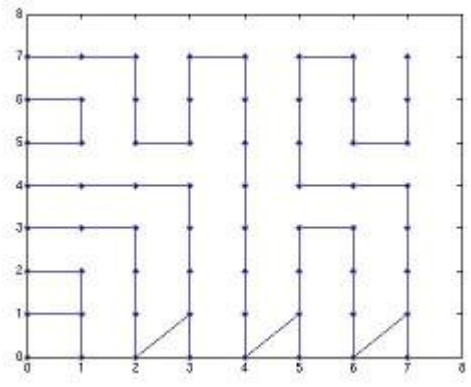

Fig. 3. $N = 3$.

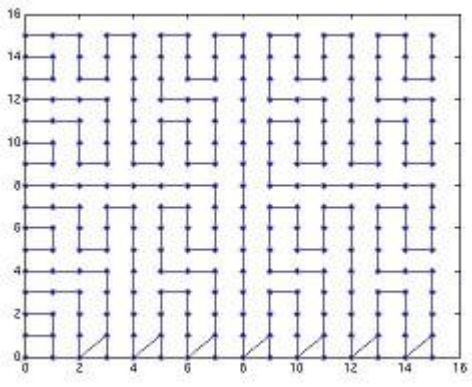

Fig.4. $N = 4$.

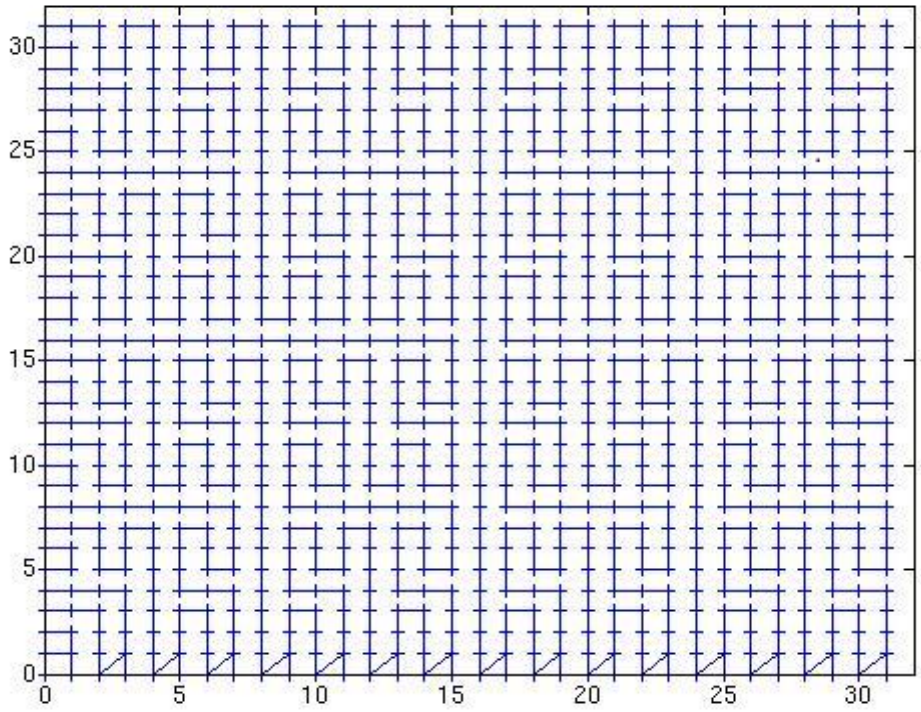

Fig. 5. $N = 5$.

Interpretation

Interpretation of Figs. 3,4,5 requires special scrutiny of several issues. To simplify the examination, the behavior near the bottom and left edges in the figures should be temporarily ignored. Some ambiguity in the notation should be tolerated, namely when $x_i = k_i 2^{-N}$, $k_i = 0,1,2, \ldots,2^N-1$. When N is understood k_i may in proper context be interpreted as a reference to $x_i = k_i 2^{-N}$, so that references such as $\boldsymbol{x} = \{k_0 , k_1\}$ and $\cdot(\{k_0 , k_1\})$ have clear meanings.

Though the graphs do not prove it, and with some exceptions and anomalies near the $x_i = 0$ borders, to be considered below and in Appendix A, all increments in Figs. 3-5 in the quantized $\boldsymbol{x}$ values involve an increment of one space in only one of the two coordinates. That was confirmed via a program written to examine that specific feature for $N = 3,4,5,6,7,8$.

The ensuing discussion makes frequent reference to Theorem 1, which relates to L0, but usually in reference to sub-squares of the main square relating to the L0 case. It is believed the reader can interpret the principles corresponding to Theorem 1 in relation to levels > 0; introduction of more notation and definitions to make the discussion literally precise would only impede its comprehension.

With those understandings, one sees points, representing quanta, joined by straight line segments that have been added by the graphing software to show how points are traversed as r increases monotonically from 0 to 1. It is necessary to distinguish between points and paths in this examination. All points here or in such depictions of any SFC represent points on the relevant SFC; it is, after all, a SFC! A notable difference between our figures and comparable figures depicting the Peano class curves is that, for pictures depicting the Peano class, the line segments do not represent segments of the SFC; the lines depicted only converge to the SFC. For the Lebesgue class (i.e. $\cdot$ and $\cdot$), by contrast, all segments along even-valued coordinates represent paths on the SFC, because they are segments of the crossed axes of some Lk case. For example, and continuing to ignore border

effects in Figs. 3-5, the horizontal or vertical line segments shown for which k_i is even represent points on the SFC ・ or ・, in accord with (1,2) and the recursive rule.

That is not the case for lines along odd-valued coordinates; they depict only the order in which points are visited as *r* passes monotonically from 0 toward 1. That is true for general *N* a power of 2. The lines along odd valued coordinates are termed "Devil's Paths" here. They do not represent segments of Lebesgue's SFC, and they all run in directions opposite to those taken by Lebesgue's SFC. The points represent values of the SFC ・・or ・, but the lines joining those points do not represent segments of the SFC, since for ・・or ・・paths > 0 in length run in opposite directions. The present discretization of ・ and ・ provides visible quantized paths > 0 in length compatible with passing from {0,0} to {1,1} for a finite binary-spaced grid, motivating the term "Devil's Paths." While neither ・ nor ・ is novel in providing new trajectories through the unit square ***X***, the pairs (・, ・) and (・, ・′) are apparently novel in providing trajectories in our class of quantized spaces (・′ is the version of ・ adjusted for use in association with ・). The concept of the "working inverse" ・, perhaps paradoxically, is essential to defining our version of calculation of a SFC on a finite binary-space grid, derived from Lebesgue's SFC It could be objected that the horizontal Devil's Paths are only one quantum in length (apart from border effects), and therefore scarcely "paths" at all, but they join the longer vertical Devil's Paths segments, making them part of structures credible as "paths". Appendix A offers a detailed examination of the trajectories shown in Figs. 3-5, including border effects.

If "border effects" are considered intolerable, they can be eliminated by eliminating the borders from the domains of interest; eliminate the coordinate 0 and let the curve run from {1,1} to $\{2^N\text{-}1,2^N\text{-}1\}$. The result of doing that, for $N = 4$, is shown in Fig. 6. The maximum length of vertical Devil's Paths is 3 quanta and 1 quantum for horizontal. Joined horizontal and vertical Devil's Path segments, excluding the Devil's Paths starting at {1,1} and those ending at {15,15}, are uniformly of length 4, a property maintained for general *N*.

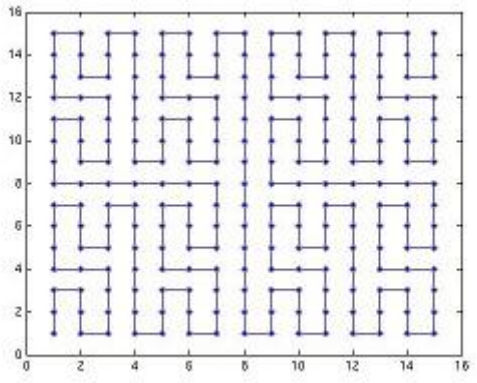

Fig. 6 Border effects eliminated on reduced grid, $N = 4$

Another feature of Fig. 6 is that the transitions from one point to another are what they appear to be in the picture. Unlike the cases treated by Figs. 3-5, every increment in the quantized ***x*** values involves an increment of one space in only one of the two coordinates; the graphing software did not hide greater increments. The double transitions at the left and bottom borders, partly hidden in Figs.3-5 (v. Appendix A), disappeared when the border points were eliminated from the computation.

The results in Figs. 3-5 (excluding borders) and Fig. 6 indicate Devil's Paths are limited in integer length to four quanta or five points regardless of the value of *N*. Thus, as *N* increases, real lengths of Devil's Paths, viewed in the continuum ***X***, approach 0, as they must, since that is their length in the continuum. No exposition of the Lebesgue SFC is directly applicable to a quantized version of the Lebesgue SFC without offering a specific solution to the problem the Devil's Paths solve. The Lebesgue SFC as defined on *R*- ・ runs only one way, indeed the wrong way.

The algorithms for computing $\cdot(r)$ and the working inverse $\cdot(\boldsymbol{x})$ could be converted so that r is represented by an integer or binary string, with multiplication and division by 8 achieved with shifts. That was the main motivation for defining $\cdot$ as an alternative to Lebesgue's $\cdot$. Fig. 6 suggests a radically different approach to such computations, eliminating any need to sort.

Fig. 6 makes the basic nature of the curve transparent and could be of help in finding a less complicated description of the SFC and its inverse than one gets using the methods treated above. Consider the sequence (Fig. 6) {1,1}→{1,2}→{1,3}→{2,3}→{2,2}→{2,1}→{3,1}→{3,2}→ {3,3}. That constellation is seen in translated form, throughout the square $\boldsymbol{X}(N)$, also in Figs. 3-5. We designate such a segment of 9 points an "N segment," for obvious reasons. It is seen that every N segment is joined, at every end (except {1,1} and $\{2^N\text{-}1, 2^N\text{-}1\}$) to an arm of some L*k* case. Peano defined his SFC algebraically, but pictorial representations of the various Peano SFCs show N segments as basic elements, though they are connected differently Sagan [30, pp. 35,44f].

An N segment can be interpreted in terms of Theorem 1, as is done for Fig. 4 in Appendix A, but a consequence of this work is to make it possible to leave the formal description of Lebesgue's SFC behind insofar as application to a binary-spaced quantized grid is concerned. It is conjectured algorithms (forward and inverse) describing the course of the version of the curve exemplified by Fig. 6, with the index r and the spatial coordinates x_0 and x_1 being represented by integers, binary strings, could be formulated directly from algebraic or algorithmic interpretations of the pictures of the Fig. 6 type, virtually ignoring Lebesgue's formulation. That is a quite satisfactory way to proceed. Hilbert defined his SFC with pictures, and thus everything said about it, since Hilbert, has proceeded from his pictures. An implicit aim of such studies would naturally be the elimination of any need to sort stored r values; such a requirement never existed for quantized versions of Peano class SFCs for which inverses have been determined. The code included here, which achieved the foregoing results, could quickly become obsolete, having served to expose a relatively simple version of Lebesgue's SFC, for a uniformly spaced binary grid, and with an inverse. The relevance of horizontal and vertical sensitivities would no longer exist, since every unit increment of the index would be reflected in a unit increment of one of the components of its image. Hybrid arrangements with the Hilbert SFC would be easily achieved.

Conclusion

The original Lebesgue space filling curve (SFC) has limited compatibility with realization with binary arithmetic on a uniformly spaced binary grid, one impediment being its formulation in terms of ternary representations; that impediment can be overcome via use of an alternative Cantor set. A second impediment, namely the Devil's Staircase feature, also created by the role of the Cantor set, can be overcome via definition of a "working inverse," thereby providing means of achieving compatibility with such a grid. The results indicate an alternative way to proceed, in realizing an approximation to Lebesgue's SFC, which circumvents any complication raised by Cantor sets, and is compatible with binary and integer arithmetic.

Conflicts of Interest

No conflicts of interest, in publishing this paper, are known.

Funding

There was no funding for this research, apart from the routine support Northwestern University grants faculty members.

Appendix A

Discussion of the trajectories of the discretized SFC

For the case *N* = 16 (Fig. 4) a few points are traced through to interpret the binary-spaced grid shown. Border effects are initially not considered, and consecutive points on the axes of some L*k* case need no explanation. Consider the point {7,2}, which is on the right arm of a L2 case; it passes to the point {7,3} in accord with Theorem 1. The point {7,4} is on the right arm of a L1 case, so {7,3} passes to {7,4} in accord with Theorem 1. A Devil's Path starts at {5,4} (L1) and passes to {5,5} according to Theorem 1. Then {5,6}, on the left are of a L2 case, is visited in accord with Theorem 1, and then {5,7} is visited in accord with Theorem 1.

Consider a few horizontal increments. The point {6,7} is on the vertical arm of a L2 case, so it is visited after {5,7} in accord with Theorem 1. The point {2,5} is on the vertical arm of a L2 case, so it passes to {3,5} in accord with Theorem 1.

A few border effects are examined. The point {4,0} is the bottom point of a L1 case so both ・({5.0}) and ・({5,1} > ・({4,0} but {5.0} and {5,1} are on the vertical arm of a L3 case with ・({5.0}) > ・({5,1}so {4,0} passes to {5,1} (an increment of two quanta), which passes to {5,0}. {5,2} is on the left arm of a L2 case so {5.0} passes to {5,2} in accord with Theorem 1. These sorts of increment of 2 quanta, along the bottom and left edges, are the only increments of 2 quanta observed. {5,2} passes to {5,3} in accord with Theorem 1 {5,3} and {6,3} are < the points on the right arm of the L1 case centered at {4,4} but {6,3} is on the vertical arm of the L2 case centered at {6,2} so by Theorem 1 {5,3} passes to {6,3}.

The behavior near {0,0} is the same for all values of *N*. The image starts at {0,0} and passes to {0,1}, viewed as being on the left arm of a L3 case. {1,2} and {0,2} are on the left arm of a L2 case, so ・({1,1}) < ・({1,2}) by Theorem 1 and {0,1} passes to {1,1}, then {1,0,} then {1,2} (increment of 2, as with the other bottom odd border points) then {0,2} along the left arm of L2 case. There are additional horizontal increments of 2 quanta on Devil's Paths on the left edges of Figs. 3-5.

Appendix B

C Code

```
/*
  program tests beta and omega computations on
a binary-spaced grid
                              */
    #define QUANT 8    /* power of 2, number of
                    vertical or horizontal lines

  QUANT by QUANT matrix will be for spacing
1/QUANT in both dimensions

  and prints the values of the index r. It is likely
these r values
  will have to be subsequently sorted if they are to
be used       */

#include "stdio.h"
#include "math.h"

#define MAX(x,y)          ((x) < (y) ? (y) : (x))
#define MIN(x,y)          ((x) < (y) ? (x) : (y))
#define RECSMAX 50 //max number of recursions
allowed
#define HALF 0.5
#define DELTA 1.0e-12

int maxrecs = 0;
double rstart = 0; //static doesn't work for this, so
it is global
int divfac = 1;
int omega_count = 0;
int recsmax_count = 0;
int error = 0;
FILE *fp;
FILE *fr;

int beta(double, double [], int);
void omega(double *, double[]);

void main()
{
  double deltax = 1.0/(QUANT);
  int k0,k1;
  int off_border = 0;
  double x[2];
  double xcall[2];
  double xret[2];
```

```
  double r;
  double err;
  double total_error = 0.0;
  double max_error = -1;
  double xmax[2] = {0.0};

  fp = fopen("grid.txt","wb");
  fr = fopen("rvalues_new.txt","w");
  printf("deltax = %f\n\n",deltax);
  fprintf(fp,"deltax = %f\n\n",deltax);
  fprintf(fr,"r values for QUANT = %d\n\n",QUANT);
  for (k0=0; k0 < QUANT; k0++) //normal use
  {
          x[0] = k0*deltax;
          for (k1=0; k1 < QUANT; k1++)// normal
use
          {
                  xcall[0] = x[0];
                  x[1]=k1*deltax;
                  xcall[1] = x[1];
                  omega(&r,xcall);
                  fprintf(fr,"%26.16lg,\n",r);
                  beta(r,xret,0);
                  err = fabs(x[0]-xret[0])+ fabs(x[1]-
xret[1]);
                  printf("%26.16lg,\n",r); //test only
                  if (err > max_error)
                  {
                          max_error = err;
                          xmax[0] = x[0];
                          xmax[1] = x[1];
                  }
                  if (error != 0)

                  {
                          if (x[0]*x[1] != 0)
                            off_border++;
                     printf("\nx[0] = %lf",x[0]);
                          fprintf(fp,"\n\nx[0] =
%20.13e",x[0]);
                          printf("   x[1] = %lf",x[1]);
                          fprintf(fp,"   x[1] =
%20.13e",x[1]);
                          printf("\n    r =
%23.18lg",r);
                          fprintf(fp,"\n    r =
%23.18lg",r);
                          printf(" confirmation x =
%20.13e,  %20.13e, \nerror = %lf\n",x[0],x[1],err);
                          fprintf(fp," confirmation x
= %210.13e,  %20.13e, \nerror =
%lf\n",x[0],x[1],err);
                          error = 0;
                  }
                          total_error += err;
                          omega_count = 0;
                  }
  }
  printf("\ntotal error= %20.12e\n",total_error);
  fprintf(fp,"\ntotal error = %20.12e\n",total_error);
  total_error=total_error/QUANT/QUANT;
  printf("\naverage error= %20.12e\n",total_error);
  fprintf(fp,"\naverage error =
%20.12e\n",total_error);
  printf("max error = %le attained for x = %lf,
%lf\n",max_error,xmax[0],xmax[1]);
  fprintf(fp,"max error = %le attained for x = %lf,
%lf\n",max_error,xmax[0],xmax[1]);
  printf("spacing of points is %le\n",deltax);
  fprintf(fp,"spacing of points is %le\n",deltax);
  printf("QUANT = %d, RECSMAX attained %d
times\n",QUANT,recsmax_count);
  printf("%d times off border\n",off_border);
  fprintf(fp,"QUANT = %d, RECSMAX attained %d
times\n",QUANT,recsmax_count);
  fprintf(fp,"%d times off border\n",off_border);
  return;
}
int beta(double r, double x[], int recs)
/*
  Computes the modified Lebesgue SFC
                                        */
{
//
  short unsigned int err;

  if (recs == RECSMAX)
  {
          printf("recursions reached the limit\n");
          fprintf(fp,"recursions reached the limit\n");
          recsmax_count++;
          return 1;
  }
  recs++;

  if (r <= 0.0)
  {
    x[0] = 0.0;
```

```
        x[1] = 0.0;
        return 0;
}
if (r >= 1.0)
{
  x[0] = 1.0;
        x[1] = 1.0;
        return 0;
}

double rx8 = 8*r;
if (rx8 < 1)
{
  err = beta(rx8,x, recs+1);
  x[0] = x[0]/2;
  x[1] = x[1]/2;
}
else if (rx8 <= 2)

{
  x[0] = MAX(0 , 1 - 4*r);
  x[1] = 0.5;
}
else if (rx8 < 3)
{
  err = beta(rx8-2,x, recs+1);
  x[0] = x[0]/2;
  x[1] = (x[1]+1)/2;
}
else if (rx8 <= 5)
{
  x[0] = 0.5;
  x[1] = MAX(2.5 - 4*r, 0);
}
else if (rx8 < 6)
{
  err = beta(rx8-5, x, recs+1);
  x[0] = (x[0]+1)/2;
  x[1] = x[1]/2;
}
else if (rx8 <= 7)
{
  x[0] = MIN(4*(1-r),1.0);
  x[1] = 0.5;
}
else
{
  err = beta(rx8-7,x, recs+1);
  x[0] = (x[0] + 1)/2;
  x[1] = (x[1] + 1)/2;
 }
 return 0;
}
void omega(double *r,double x[])
/*
    Computes the working inverse
                                */
{
   omega_count++;
   if (x[0] <= 0.0 && x[1] <= 0.0)
   {
        *r = 0;
        return;
        }
   if (x[0] >= 1.0 && x[1] >= 1.0)
   {
        *r = 1.0;
        return;
        }
        if (fabs(x[0]-HALF) < DELTA)
    x[0] = HALF;
   if (fabs(x[1]-HALF) < DELTA)
    x[1] = HALF;
        if ((x[0] == HALF) || (x[1] == HALF))
        {
          if (x[0] == HALF)
          {
            *r = (2.5-x[1])/4;
          }
          else
          {
                if (x[0] < HALF)
                  *r = (1-x[0])/4;
                else

                  *r = 1 - x[0]/4;
          }
    *r = *r/divfac + rstart;
    rstart = 0;
    divfac = 1;
    return;
        }
        else
        {
                divfac = 8*divfac;
                x[0] = 2*x[0];
                if (x[0] > 1)
                {
```

```
                x[0] -= 1;
                rstart += 5.0/divfac;
            }
            x[1] = 2*x[1];
            if (x[1] > 1)

            {
                x[1] -= 1;
                rstart += 2.0/divfac;
            }
            omega(r,x);
        }
}


/* based on sorting program from
https://stackoverflow.com/questions/20584499/why-qsort-from-stdlib-doesnt-work-with-double-values-c
*/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "rvalues.h"

/* if you are told that the compare statement must
   be preceded with ',' or ';' then you probably
forgot
   to terminate the header rvalues.h with a
semicolon  */
int compare (const void * a, const void * b)
{
   if (*(double*)a > *(double*)b) return 1;
   else if (*(double*)a < *(double*)b) return -1;
   else return 0;
}

int main()
{
        FILE *dfp;
   int n;

/*    printf("Before sorting the list is: \n");
   for( n = 0 ; n < 5; n++ )
   {
      printf("%.2f ", values[n]);
   }

   printf("\n\n");    */

   if ((dfp = fopen("rsorted.txt","wb")) == NULL)
     printf("Can't open output for writing\n");
          else
     qsort(r, LENGTH, sizeof(double), compare);

   printf("\nAfter sorting the list is: \n");
   for( n = 0 ; n < LENGTH; n++ )
   {
      printf("%    20.12e,\n", r[n]);
      fprintf(dfp,"%20.12e,\n",r[n]);
   }

   printf("\n\n");
   fclose(dfp);
   return(0);
}
```

example header file for sorting program.

```
// rvalues.h
//unordered r values for QUANT = 8 for sorting
program
#define LENGTH 64 // = QUANTxQUANT
double r[LENGTH] = {
                    0,
              0.00390625,
                0.03125,
              0.03515625,
                  0.25,
              0.25390625,
                0.28125,
              0.28515625,
             0.009765625,
               0.0078125,
               0.0234375,
               0.0390625,
                0.21875,
               0.2578125,
               0.2734375,
               0.2890625,
                0.078125,
               0.0703125,
                0.0625,
               0.0546875,
                0.1875,
               0.3203125,
                0.3125,
               0.3046875,
```

```
0.087890625,
0.0859375,
0.1015625,
0.1171875,
0.15625,
0.3359375,
0.3515625,
0.3671875,
0.625,
0.59375,
0.5625,
0.53125,
0.5,
0.46875,
0.4375,
0.40625,
0.634765625,
0.6328125,
0.6484375,
0.6640625,
0.84375,
0.8828125,
0.8984375,
0.9140625,
0.703125,
0.6953125,
0.6875,
0.6796875,
0.8125,
0.9453125,
0.9375,
0.9296875,
0.712890625,
0.7109375,
0.7265625,
0.7421875,
0.78125,
0.9609375,
0.9765625,
0.9921875,
};
```

References

1. G. Peano. Sur une courbe, qui remplit toute une aire plane. *Math. Ann.*, 36(1):157-160, 1890. https://doi.org/10.1007/BF01199438.

2. D. Hilbert. Über die stetige Abbildung einer Linie auf ein Flächenstück. *Math. An.*,38(3): 459-460, 1891.

3. A. Bos and H. Haverkort, "Hyperorthogonal well-folded Hilbert curves," *J. Computational Geometry*," **7**, 2, 145-190, 2016.

4. A.R. Butz, "Space filing curves and mathematical programming," *Information and Control*, **12**, 4, 314-330, April 1968.

5. A.R. Butz, "Convergence with Hilbert's space filling curve," *J. Computer and System Sciences*, **3**, 128-146, 1969.

6. A.R. Butz, "Alternative algorithm for Hilbert's space-filling curve," *IEEE Trans. Computers*, April 1971, 424-426.

7. M. Bailova, J. Bouchala, and Petr Vodstrcil, "Global Optimization Using Space Filling Curves," *Math. Analysis and Numerical Mathematics*, **15**, 2, June 2017.

8. D. Lera1 and Y. D. Sergeyev, GOSH: derivative-free global optimization using multi-dimensional space-filling curves, J Glob Optim Nov. 2017.

9. C. Schretter, Z. He, M. Gerber, N. Chopin, H. Niederreiter "Van der Corput and Golden Ratio Sequences Along the Hilbert Space-Filling Curve" Monte Carlo and Quasi-Monte Carlo Methods pp 531-544, online June 2016. https://link.springer.com/chapter/10.1007/978-3-319-33507-0_28.

10. Y.D. Sergeyev, R.G. Strongin and D. Lera, *Introduction to Global Optimization Exploiting Space-Filling Curves*, Springer, 2013.

11. A. Obulesu, V.V. Kumar, and L. Sumalatha, "Image Retrieval Based on Local Motif Patterns Code, *Int. J. Image, Graphics and Signal Processing,* 2018, 6, 68-78.

12. A.M. Reddy, V.V. Krishna and I. Sumalatha, "Face Recognition based on Cross Diagonal Complete Motif Matrix," *I.J. Image, Graphics and Signal Processing,* 2018, 3, 59-66. Uses "Peano scan".

13. Lin Fu, "Numerical methods for computational fluid dynamics - a new ENO paradigm and a new domain decomposition method", dissertation, Technischen Universität München, Feb. 10, 2017.

14. M. Bader, *Space-Filling Curves*, Springer, Berlin, 2013. Bader's book cites many examples of applications.

15. L. Arge, M de Berg, H. Haverkort, and K Yi, "The priority R-tree: a practically efficient and worst-case optimal R-tree." *ACM Trans. Algorithms*,**4**(1), 30 (2008). Art. 9.

16. K. Gao and X. Mao, "Research on Massive Tile Data Management based on Hadoop," 2016 2nd International Conference on Information Management (ICIM), London.

17. X. Guan, P. van Oostrom, and B. Cheng, "A Parallel N-Dimensional Space-Filling CurveLibrary and Its Application in Massive Point Cloud Management," *Int. J. Geo-Information*, 2018, **7**, 327+. Supports libraries using the Hilbert SFC and the Morton order

18. I. Kamel and C. Faloutsos, "Hilbert R-tree. An improved R-tree using fractals," Proc. 20th VLDB Conf., Santiago, Chile, 1994. 500-509.

19. I. Kamel, and C. Faloutsos. "On packing R-trees," Conf. on Information and KnowledgeManagement, ACM, 1993, 490-499, 1993, 500-509.

20. J. Lawder, "The Application of Space-Filling Curves to the Storage and Retrieval of Multi-dimensional Data," Ph.D. Thesis, Univ. of London, Dec.1999.

21. J. Banerjee, R. Ghatak, and A. Karmakar, "A compact planar UWB MIMO diversity antenna with Hilbert fractal neutralization line for isolation improvement and dual band notch characteristics," Conference: 2018 Emerging Trends in Electronic Devices and Computational Techniques (EDCT), March 2018. DOI: 10.1109/EDCT.2018.8405070.

22. H. Liu et. al. , "Dynamic Load Balancing Using Hilbert Space-Filling Curves for Parallel Reservoir Simulations" Society of Petroleum Engineers, SPE Reservoir Simulation Conference, 20-22 February, Montgomery, Texas, USA, 2017 (https://www.onepetro.org/conference-paper/SPE-182613-MS).

23. L.N. Kanal, "Interactive pattern analysis and classification systems: a survey and commentary," *Proc. IEEE*, vol. 60, no. 10, Oct. 1972, 1200-1215.

24. K. Davis and Y. Li, "Efficient 3D Inversion of Magnetic Data Via Octree Mesh Discretization, Space-filling Curves, And Wavelets." Society of Exploration Geophysicists, 2010-1172 SEG Conference Paper – 2010, Jan. 1, 2010.

25. F. Ancona, A. De Gloria and R. Zunino, "Parallel VLSI architectures for cryptographic systems," conference, 1997.

26. D.J. Hancock et. al. "An Investigation of Feedback Guided Dynamic Scheduling of Nested Loops," 2000, Parallel Processing, 2000. Proceedings. 2000 International Workshops on, 21-24 Aug. 2000, Toronto. IEEE.

27. C. Muelder and K-L Ma, "Rapid Graph Layout Using Space Filling Curves," *IEEE Trans. Visualization and Computer Graphics*, **14**, 6 Nov./Dec. 2008.

28. W.D. Blecher, The Hilbert-Moore sequence Acoustic Noise optimized MR Imaging, dissertation, Universität Mannheim, 2008.

29. D. Moore. "Fast Hilbert curve generation, sorting, and range queries". http://www.tiac.net/~sw/2008/10/Hilbert/moore/, 2000.

30. H. Sagan, *Space-Filling Curves*, Springer, Springer Science+Business Media, New York, 1994.

31. H. Haverkort, "Recursive Tilings and Space-Filling Curves with Little Fragmentation." *J. Computational Geometry*, vol. 2, no. 1, 92-127, 2011.

32. H. Lebesgue, *Leçons sur l'intégration*, Gauthier-Villars, Paris, 1904 and 1928, (pp 44f). H. Lebesgue, "Sur les fonctions représentables analytiquement," *Journal de Mathématique Pures et Appliquées*, vol. 6, no. 1 (1905), 139-216 (pp. 210f).