A Generalized 2D and 3D Hilbert Curve

Jakub Červený and Zzyv Zzyzek

Abstract

Hilbert curves are classic space-filling curves with strong locality properties, widely used for linearizing 2D and 3D data. In their discrete form, however, the standard Hilbert construction applies most naturally to square or cubic grids with side lengths that are exact powers of two. We present the *Gilbert curve*, a simple recursive, Hilbert-like construction that produces Hamiltonian paths on *arbitrary* 2D and *even-sized* 3D axis-aligned rectangular grids. Our construction explicitly manages endpoint parity constraints, and extends naturally to 3D.

1 Introduction

Space-filling curves map a one-dimensional ordering to a multi-dimensional grid while preserving locality as much as possible. Discrete Hilbert curves, in particular, are commonly used to linearize images and volumes for cache-coherent traversal, spatial indexing, data layout, and more. In practical settings, however, the underlying grids are rarely square or cubic with power-of-two side lengths. Images and volumes come in arbitrary dimensions; tiling, padding, or resampling to fit a power-of-two Hilbert curve introduces overhead and can harm locality near boundaries.

This paper presents the *Gilbert curve*, a recursive construction that produces a Hamiltonian path (a non-self-intersecting path visiting every grid vertex exactly once) on arbitrary 2D rectangles and 3D cuboids. Our goal is a construction that is easy to implement, has a uniform recursive structure, and reduces to the standard Hilbert curve in the power-of-two square/cube cases.

Contributions.

- We describe a simple recursive construction of Hilbert-like Hamiltonian paths on arbitrary 2D rectangular grids.
- We formalize a parity ("color compatibility") condition on corner endpoints and use it to guide valid recursive subdivisions.
- We extend the same construction principles naturally to arbitrary 3D cuboid grids, reusing the 2D routine for planar subproblems.

2 Related Work

Hilbert introduced his eponymous curve as a continuous mapping from the unit interval to the unit square [3]. Many discrete variants exist; the most common recursive discretization applies most directly to power-of-two square (2D) and cubic (3D) grids.

Tautenhahn [5] presents an unpublished 2D construction for arbitrary sizes based on combining 3×3 Peano and 2×2 Hilbert block types. While that approach is 2D-only and not purely uniform across scales, it motivates an important endpoint-parity constraint that we make explicit in Section 4.

Zhang, Kamata, and Ueshige [6] propose a pseudo-Hilbert scan for arbitrary rectangles. Compared to our construction, their method follows a more intricate case structure and does not suggest a direct extension to 3D.

3 Examples

Figure 1 shows 2D Gilbert curves on a square grid, an eccentric rectangle, and a rectangle with a small notch. Figure 2 shows analogous 3D examples.

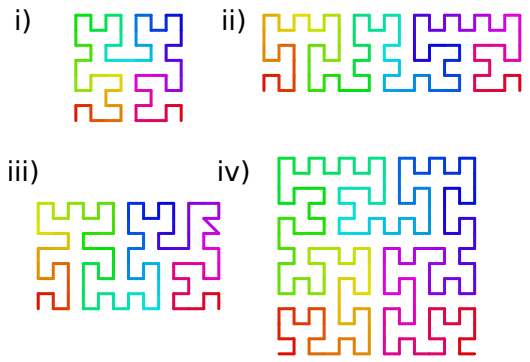


Figure 1: *2D Gilbert curves for i)* 8×8*, ii)* 18×6*, iii)* 13×8 *(with notch), iv)* 14×14.

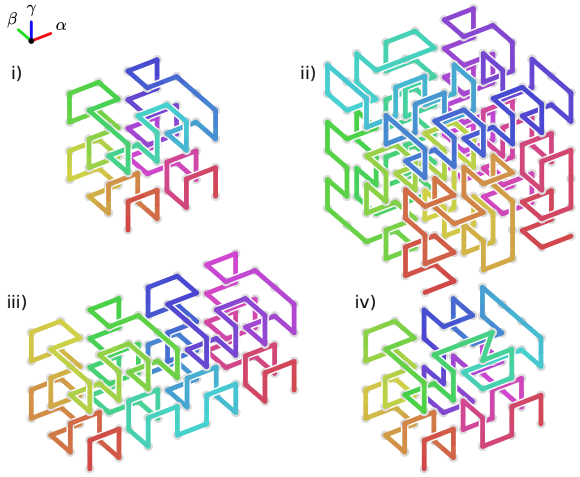
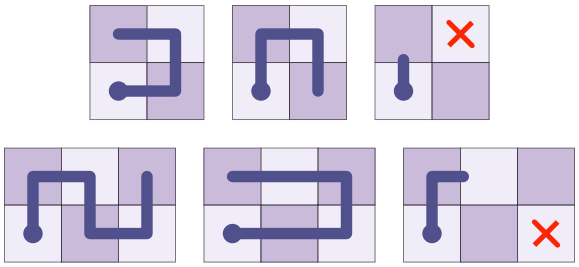


Figure 2: *3D Gilbert curves for i)* $4 \times 4 \times 4$*, ii)* $6 \times 6 \times 6$*, iii)* $8 \times 4 \times 4$*, iv)* $5 \times 4 \times 4$ *(with notch).*



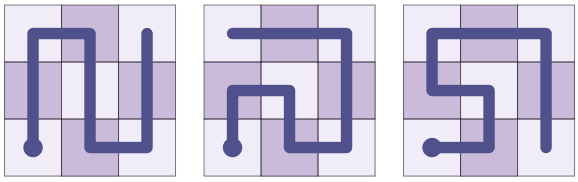


Figure 3: *Hamiltonian path feasibility for small grids. A red 'x' marks a corner endpoint that is impossible given the chosen start.*

4 Parity constraints on corner endpoints

We will repeatedly need to know when a Hamiltonian path is even possible between two corners of a grid. The key constraint is parity. Consider the standard checkerboard coloring of an $m \times n$ grid graph (or an $m \times n \times p$ grid graph in 3D): adjacent vertices always have opposite color. Any path alternates colors at every step, so the parity of the endpoints is constrained by the total number of visited vertices.

Lemma 1 (Color compatibility)**.** *Let G be a rectangular 2D grid graph with mn vertices, or a rectangular 3D grid graph with mnp vertices. Fix a start corner s and an end corner t. If there exists a Hamiltonian path from s to t, then s and t have the* same *color when the number of vertices is odd, and* opposite *colors when the number of vertices is even.*

Proof. Along any path, vertex colors alternate. A Hamiltonian path visits all N vertices exactly once and therefore has $N-1$ steps. If N is odd, then $N-1$ is even and the endpoints must have the same color; if N is even, then $N-1$ is odd and the endpoints must have opposite colors. □

We call a pair of endpoints *color compatible* if they satisfy the lemma. Figure 3 illustrates the constraint for small 2D grids. Color compatibility is necessary in general; for the specific family of corner-to-corner subproblems produced by our construction, it is also sufficient, and we use it as a lightweight validity check when selecting subdivisions.

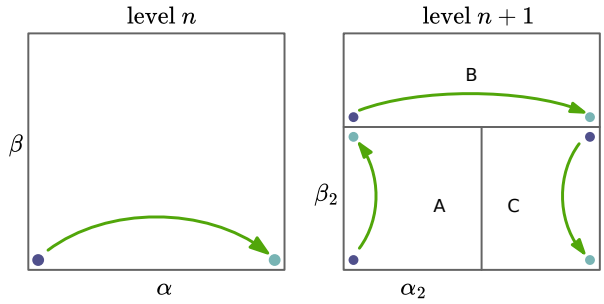


Figure 4: *Subdivision template for the 2D Gilbert curve.*

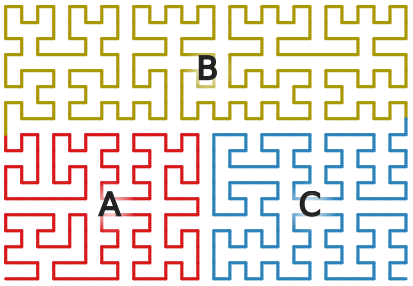


Figure 5: *Example curve with subdivision regions highlighted.*

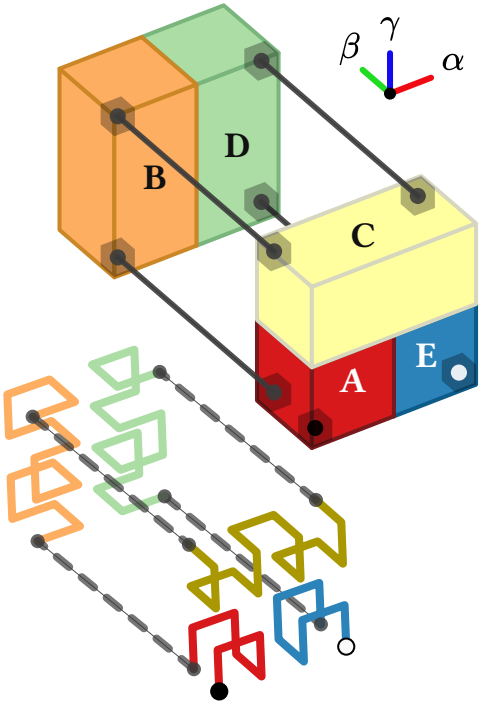


Figure 6: *The main subdivision template for the 3D Gilbert curve.*

5 The 2D Gilbert curve

Our 2D construction recursively partitions an $m \times n$ rectangle into a small number of axis-aligned subrectangles and concatenates the subpaths into a single Hamiltonian path. Figure 4 shows the main 2D subdivision template and Figure 5 illustrates one recursion with subregions highlighted.

The vectors $\alpha, \beta, \gamma \in \mathbb{Z}^3$ provide generalized notions of width (α), height (β), and depth (γ), so that we can transform rectangular or cuboid regions as necessary for the recursion. Each of α, β, γ will only have one non-zero component and will be orthogonal to each other, representing a snapshot of the current frame of reference.

When dividing into subregions, we choose integral side lengths and prefer even lengths where possible to satisfy the parity constraints of Section 4. Each subregion is solved recursively and then stitched to its neighbors. Figure 7 summarizes the main parity-driven cases.

When one side length is much larger than the other, we optionally apply an *eccentric split* that first bisects the long side. This improves robustness on highly elongated rectangles and keeps the recursion structure

simple.

2D Gilbert Curve

For the 2D Gilbert curve, the region is subdivided into two squares at the bottom and a rectangle on top, with endpoints joined on the outer perimeter. This process is recursively applied until a trivial path can be produced. Figures 4 and 10 show two different representations of the template used for the subdivision. Figures 5 and 11 show color coded examples for dimensions 26×18 and 42×30 respectively.

If the width like dimension (α) is much larger than the height like dimension (β), an alternate eccentric subdivision template is used to split the region in half along the α length (figure 8, 9).

Each regions parity is chosen so as to try and keep color compatibility, ensuring a valid path if possible. If the α axis line that the endpoints fall on is odd, with the orthogonal β direction even, a notch is forced and is directed to the upper right region, as illustrated by the "even/odd" case in figure 7.

When the side lengths are exact powers of two, the resulting curve is identical to the 2D Hilbert curve.

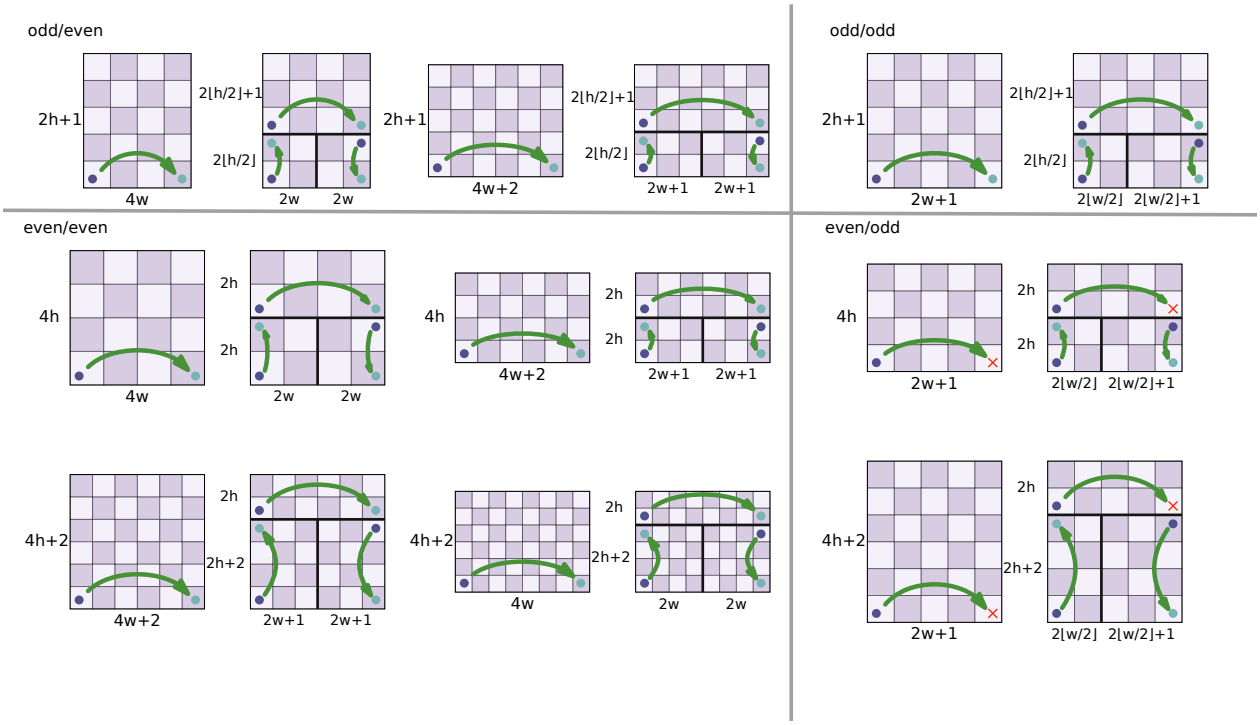


Figure 7: *Enumeration of the subdivision template depending on different parities of α and β dimensions.*

6 Extension to 3D

In the normal case, a subdivision scheme is used to split the cuboid into five regions. Two cube like regions are partitioned where the path starts and stops, and three oblong cuboid regions are partitioned for the middle portion of the path. Figure 6 provides an exploded view of the main subdivision and where the endpoints line up. See also figure 18 for the template and Figure 19 for a $10 \times 10 \times 10$ example, with each region color coded.

Endpoints within a sub-divided region are kept on the exterior of the parent cuboid and joined after the resulting recursion has completed. As with the 2D Gilbert curve, when side dimensions are equal and exact powers of two, the resulting curve is identical to the corresponding 3D Hilbert curve.

During the course of sub-division, if the width like dimension (α) is much larger, the curve is split in half along the α length (Figure 12, 13). If the α eccentric split is not triggered, the height-like dimension, β, is then compared to the depth-like dimension, γ and, if very much larger, an eccentric subdivision scheme is used that splits in the α and β direction (Figure 14, 15). If both α and β are not much larger, a check is done to see if the depth-like dimension, γ, is much larger than than the height-like dimension, β, splitting in the α and γ direction if so (Figure 16, 17).

If any of the subdivided regions has endpoints that lay in a direction with odd length, a notch will appear. This means the number of notches for the 3D Gilbert curve isn't limited to one in the case a side length is odd.

7 Algorithms

Algorithms 1 and 2 give pseudocode for the 2D and 3D constructions. We represent an oriented grid-aligned rectangle or cuboid by an origin $p \in \mathbb{Z}^3$ and axis vectors $\alpha, \beta, \gamma \in \mathbb{Z}^3$ whose nonzero components encode the side lengths and directions. The function $\delta(\cdot)$ returns the unit axis direction of its argument, and $\mathrm{div}(\cdot,\cdot)$ denotes integer division (rounding toward zero). Eccentric splits are triggered by simple aspect-ratio thresholds in the pseudocode.

Algorithm 1 Gilbert 2D

```
# p, α, β ∈ ℤ³
function GILBERT2D(p, α, β)
    α₂, β₂ = div(α, 2), div(β, 2)
    if (|β| ≡ 1) then
        yield p + i · δ(α) forall i ∈ |α|
    else if (|α| ≡ 1) then
        yield p + i · δ(β) forall i ∈ |β|
    else if (2|α| > 3|β|) then
        if (|α₂| > 2) and
           (|α₂| mod 2 ≡ 1) then
            α₂ ← α₂ + δ(α)
        end if
        yield Gilbert2D(p, α₂, β)
        yield Gilbert2D(p + α₂, α − α₂, β)
    else
        if (|β₂| > 2) and
           (|β₂| mod 2 ≡ 1) then
            β₂ ← β₂ + δ(β)
        end if
        yield Gilbert2D(p,
                        β₂, α₂)
        yield Gilbert2D(p + β₂,
                        α, (β − β₂))
        yield Gilbert2D(p + (α − δ(α))+
                        (β₂ − δ(β)),
                        −β₂, −(α − α₂))
    end if
end function
```

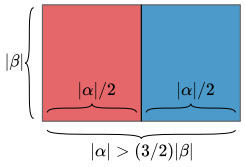


Figure 8

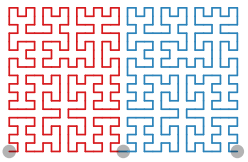


Figure 9

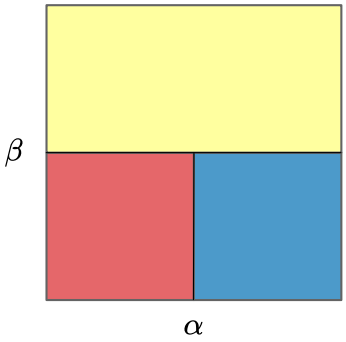


Figure 10

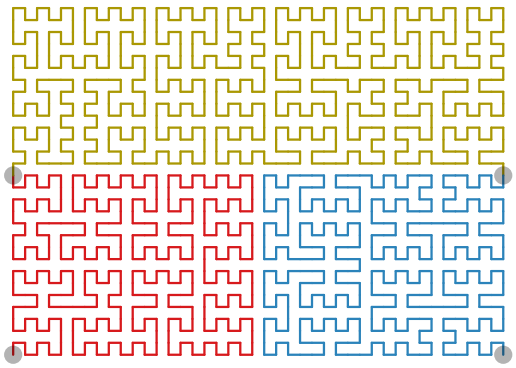


Figure 11

Algorithm 2 Gilbert 3D

```
# p, α, β, γ ∈ ℤ^3
function GILBERT3D(p, α, β, γ)
    return GILBERT2D(p,β,γ) if (|α| ≡ 1)
    return GILBERT2D(p,α,γ) if (|β| ≡ 1)
    return GILBERT2D(p,α,β) if (|γ| ≡ 1)
    α2 ← div(α, 2) + Δ(α2, α)
    β2 ← div(β, 2) + Δ(β2, β)
    γ2 ← div(γ, 2) + Δ(γ2, γ)
    if (2|α| > 3|β|) and (2|α| > 3|γ|))
        yield GILBERT3D(p,α2,β,γ)
        yield GILBERT3D(p + α2,α − α2,β,γ)
    else if (3|β| > 4|γ|)
        yield GILBERT3D(p,β2,γ,α2)
        yield GILBERT3D(p + β2,α,β − β2,γ)
        yield GILBERT3D(p+
                        (α − δ(α))+
                        (β2 − δ(β)),
                        −β2,γ,−(α − α2))
    else if (3|γ| > 4|β|)
        yield GILBERT3D(p,γ2,α2,β)
        yield GILBERT3D(p + γ2,α, β, γ − γ2)
        yield GILBERT3D(p+
                        (α − δ(α))
                        (γ2 − δ(γ)),
                         −γ2,−(α − α2), β)

    else
        yield GILBERT3D(p,β2,γ2,α2)
        yield GILBERT3D(p + β2,γ,α2,(β − β2))
        yield GILBERT3D(p+
                        (β2 − δ(β))+
                        (γ − δ(γ)),
                         α,−β2,−(γ − γ2))
        yield GILBERT3D(p+
                        (α2 − δ(α))+
                        β2 + (γ − δ(γ)),
                         −γ,−(α − α2),(β − β2))
        yield GILBERT3D(p+
                        (α − δ(α))+
                        (β2 − δ(β)),
                         −β2,γ2,−(α − α2))
end function
```

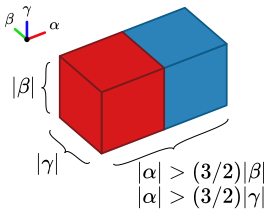


Figure 12

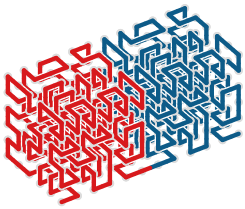


Figure 13

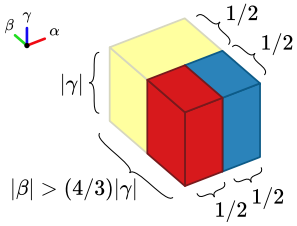


Figure 14

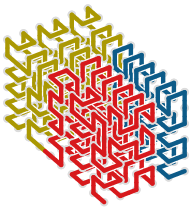


Figure 15

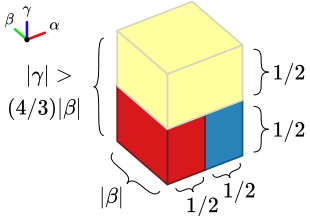


Figure 16

Figure 17

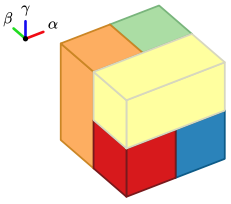


Figure 18

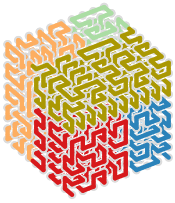


Figure 19

8 Discussion and Outlook

The Gilbert curve provides a practical alternative to discrete Hilbert curves when the target grid has arbitrary rectangular dimensions. This avoids padding or resampling that is otherwise required to fit power-of-two constructions, while preserving a recursive, locality-preserving traversal. Such layouts are useful for cache-coherent image/volume traversal, spatial indexing, and visualization of 1D data on 2D/3D grids [1, 2, 4].

There are several natural directions for future work. First, while our eccentric-split thresholds are simple and work well in practice, one can define more explicit optimization criteria for when and how to switch subdivision templates. Second, in 3D, relaxing subproblem endpoint constraints may allow constructions that further reduce the occurrence of forced diagonal steps when one or more side lengths are odd. Finally, more refined choices of subdivision could be explored focused on regularity metrics and/or perceptual quality.

9 Conclusion

We presented the Gilbert curve, a simple recursive construction of Hilbert-like Hamiltonian paths on arbitrary 2D rectangles and 3D cuboids. The construction is guided by a clean parity (color compatibility) constraint on corner endpoints and extends naturally from 2D to 3D.

A libre/free/open implementation for the Gilbert curve in 2D and 3D has been developed and can be downloaded from its repository [1].

References

[1] S. Anders. "Visualization of genomic data with the Hilbert curve." *Bioinformatics*, vol. 25, no. 10, pp. 1231-1235, 2009. https://doi.org/10.1093/bioinformatics/btp152

[2] A. Cortesi. *Visualizing binaries with space-filling curves*. 2011. https://corte.si/posts/visualisation/binvis/.

[3] D. Hilbert. "Ueber die stetige Abbildung einer Linie auf ein Flächenstück.", *Mathematische Annalen*, vol. 38, 1891, pp. 459-460

[4] S. Lee, K. Ma. "HINTs: Sensemaking on large collections of documents with **H**ypergraph vizualization and **INT**elligent agents." *IEEE Transactions on Visualization and Computer Graphics*, vol. 31, no. 9, 2025, pp. 5532-5546

[5] L. Tautenhahn. *Draw a space-filling curve of arbitrary size*, 2003. https://lutanho.net/pic2html/draw_sfc.html.

[6] J. Zhang, S. Kamata, and Y. Ueshige. "A Pseudo-Hilbert Scan Algorithm for Arbitrarily-Sized Rectangle Region." *Advances in Machine Vision, Image Processing, and Pattern Analysis*, Berlin, Heidelberg, 2006, pp. 290-299.

[1] https://github.com/jakubcerveny/gilbert