A Generalized 2D and 3D Hilbert Curve

Jakub Červený, Zzyv Zzyzek

Abstract

The two and three dimensional Hilbert curves are fractal space filling curves that map the unit interval to the unit square or unit cube. Hilbert curves preserve locality, keeping distance from the unit interval to their respective higher dimensional embeddings. Finite approximations of the Hilbert curve can be constructed in stages by stopping the recursive construction at a specified depth, providing locality from a discrete array of points to points in the embedded dimension. The locality property of the Hilbert curve can help with caching optimizations based on order and finds uses as a companion method in applications ranging from job scheduling to scene rendering. One limitation of the Hilbert curve approximation construction is that the regions need to be exact powers of two, making it difficult to work with in many real world scenarios. In this paper, we present the Gilbert curve, a conceptually straight forward generalization of the Hilbert curve, that works on arbitrary rectangular regions. The construction provides reasonable worst case run-time guarantees for random access lookups for both two and three dimensions. We also provide comparisons to other Hilbert curve generalization methods and investigate overall quality metrics of the Gilbert curve construction.

1. Introduction

1.1. Overview

We present the *Gilbert curve*, a generalized Hilbert curve for 2D and 3D, that works on arbitrary rectangular regions.

An overview of the benefits of the Gilbert curve are that it is:

- Valid on arbitrary rectangular regions
- Equivalent to the Hilbert curve when dimensions are exact powers of 2
- Efficient at random access lookups ($O(\lg N)$)
- A conceptually straight forward algorithm
- Able to realize curves without notches unnecessarily and limits the resulting curve to a single notch when forced
- Similar in measures of locality to the Hilbert curve

Some drawbacks are that:

- Our implementation is recursive, which may be undesirable for some applications requiring better than $O(\lg N)$ memory usage or better constant bounds for runtime performance
- Might not adequately capture some features of a Hilbert curve

Further, we show:

- Measures of locality are preserved (Section X)
- Trivial extensions to create generalized Moore curves (Section X)
- Comparison metrics to other space filling curves (Section X)

Generalizations of the Hilbert curve to non power of two rectangular cuboid regions have been explored before but are overly complicated algorithmically, create unbalanced curves and often don't generalize well to 3D. Our realization focuses on a conceptually simple algorithm which creates pleasing resulting curves and works in both 2D and 3D.

Space filling curves are a specialization of a more general Hamiltonian path, but have a more stringent requirement of local connectivity. The local connectivity, or locality, preserves a measure of nearness, where points from the source unit line remain close in the embedded space.

The local connectivity requirements preclude things like *zig-zag* Hamiltonian paths, as nearby points in the embedded dimension can be far from the origin line.

The Gilbert curve algorithm works by choosing sub rectangular cuboid regions, or blocks, to recursively find a connecting path. If one extent of the cuboid becomes too large or small relative to the other lengths, a special subdivision is performed to try and make further subdivisions more cube-like. Subdivisions are performed until a base case is reached and the lowest unit of the curve can be realized.

A Hamiltonian path is not always realizable for certain side lengths and endpoint constraints. In such a case, the Gilbert curve will introduce a single diagonal path move, called a *notch*.

1.2. Definitions

We concern ourselves with a space filling curve, $\omega = (\omega_0, \omega_1, \ldots, \omega_{N-1})$, through a rectangular region (N_x, N_y, N_z), $N = (N_x \cdot N_y \cdot N_z)$.

$$\begin{aligned} k \in & \quad \{0 \ldots (N-1)\}, \\ & \quad x_k, y_k, z_k \in \mathbb{N} \\ \omega_k = & \quad (x_k, y_k, z_k) \\ \omega = & \quad (\omega_0, \omega_1, \ldots, \omega_{N-1}) \\ k > 1 & \quad \rightarrow |\omega_{k-1} - \omega_k| = 1 \\ \forall i, j \in & \quad \{0 \ldots (N-1)\}, i \neq j \\ \rightarrow & \quad \omega_i \neq \omega_j \end{aligned}$$

For convenience, we use curves in 3D with the understanding that the 2D case is a specialization when $N_z = 1$ and drop the third dimension when convenient and context is clear.

A *Hamiltonian path* is a path through a graph such that every vertex is visited exactly once. In the context of this paper, we concern ourselves with Hamiltonian paths through the implied graph from integral points in 3D restricted to a cuboid region (N_x, N_y, N_z). Each point in the cuboid represents a vertex in the implied graph, with vertices joined in the graph if points are (Euclidean) distance one away from a neighboring point restricted to the cuboid.

We also concern ourselves with Hamiltonian paths that specify a *start* and *end* point within a cuboid region.

TODO

- Define locality/local connectivity
- Define space filling

https://github.com/jakubcerveny (J. Červený)
0009-0005-2175-1166 (Z. Zzyzek)

- Define admissible/feasible for Hamiltonian path?
- Define "eccentric" (oblong?)

2. Related Work

3. Algorithm

3.1. Overview

We discuss one possible generalization of the Hilbert curve, which we call the Gilbert curve, to arbitrary axis-aligned cuboid regions.

The design of the Gilbert curve is done to provide a:

- Conceptually simple algorithm
- Harmonious realization
- Resulting curve with good locality conditions

The discretized Hilbert curve recursively traces out a Hamiltonian path through a square region whose sides are powers of two. The Gilbert curve extends this idea to trace out Hamiltonian paths through to arbitrary axis-aligned cuboid regions.

The Gilbert curve recursively subdivides a larger cuboid region into smaller cuboid regions of different sizes and orientations. The subdivision scheme uses a shape template that will be explained in more detail in subsequent sections. The orientations of the subdivided cuboid regions are chosen so that the recursive path will start and end at pre-specified endpoints, connecting neighboring cuboid regions after a path is realized.

If a cuboid becomes too oblong, or *eccentric*, a subdivision shape template scheme is chosen in an attempt to create subdivisions that are closer to being cube like.

The subdivided cuboid region is processed by the Gilbert curve algorithm, with virtual origin point $p \in \mathbb{Z}^3$ and local coordinate system $[\alpha, \beta, \gamma]$, with $\alpha, \beta, \gamma \in \mathbb{Z}^3$. Each of α, β and γ are axis aligned and orthogonal.

In a cuboid region, the the Hamiltonian path start at p and ends at $p + \alpha$. For this reason, we call α the "width-like" dimension, with β and γ called the "height-like" dimension and "depth-like" dimension, respectively.

When processing the subdivided cuboid regions, the coordinate system is updated with new integral lengths and rotated. Since rotations are only in units of $(\pi/2)$ radians and α, β, γ start off as axis-aligned and orthogonal, they remain axis aligned and orthogonal throughout.

When applying the shape template to choose the subdivided cuboid regions during the coarse of the algorithm, side lengths are chosen so as to preserve the possibility of a Hamiltonian path. Should the lengths of the cuboid being subdivided preclude a Hamiltonian path from occurring, the lengths of the subdivided cuboids are chosen so that all but one will preserve the possibility of a Hamiltonian path. In this case, since only a single cuboid absorbs the impossibility of a Hamiltonian path, and this process is recursively applied, there will be a single jump, or *notch*, in the resulting curve, globally.

The next sub-section discusses parity arguments for when a valid curve can be traced out in a sub-region. We then discuss in detailed the 2D Gilbert curve algorithm and end with a discussion of the 3D Gilbert curve.

When talking about the 2D Gilbert curves, we assume vectors are in $\mathbb{Z}^3$ so they can be used without alteration for algorithms working in 3D.

3.2. Valid Paths from Grid Parity

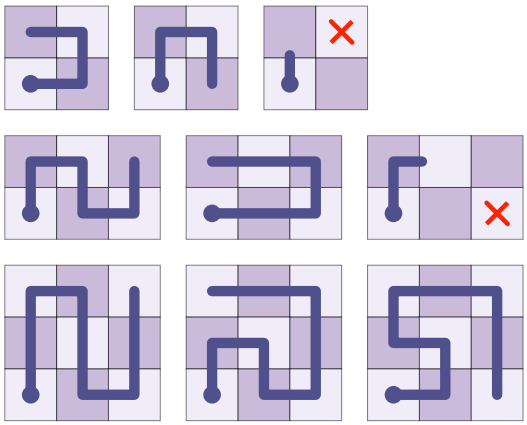

Figure 1: Illustrative examples of Hamiltonian paths height/width that are even/even, even/odd and odd/odd, respectively, when starting from the lower left hand corner

The feasibility of determining whether there exists a Hamiltonian path in a rectangular cuboid grid region can be accomplished through parity arguments. Label grid cell points in a volume as 0 or 1, alternating between labels with every axis-aligned single step move. Any Hamiltonian path that ends at one of the three remaining corners has to have the same parity as the starting point if the volume is odd, or different parity if the volume is even.

Path Possible		Volume	
		even	*odd*
$\lvert\alpha\rvert$	*even*	Yes	Yes
	odd	**No**	Yes

Table 1
Table showing when a Hamiltonian path is possible. Here, $|\alpha|$ is the absolute difference in start and end position of the path which coincides with one of the axis aligned side length of the cuboid volume. A Hamiltonian path is not possible on only in the case when $|\alpha|$ is odd and the volume is even.

For a path starting at $(0, 0, 0)$ and ending $|\alpha|$ steps in one of the axis-aligned dimensions, then Table 1 enumerates this condition under which a valid path is possible. Figure 1 illustrates this for starting position $(0, 0)$ with volumes (2×2), (3×2) and (3×3), where a red cross indicating a precluded endpoint.

Without loss of generality, we will assume a curve starts from position $p_s = (0, 0, 0)$ and has proposed endpoint at $p_e = ((w - 1), 0, 0)$, with a cuboid region as $\alpha = (w, 0, 0), \beta = (0, h, 0), \gamma = (0, 0, d)$. We state, without proof, that a Hamiltonian path is always possible from p_s to p_e when $|\alpha|$ is even or when $|\alpha|$, $|\beta|$ and $|\gamma|$ are all odd $(|\alpha| \cdot (1 - |\beta| \cdot |\gamma|) \equiv 0 \bmod 2)$.

With this condition, any cuboid subdivision will always have a Hamiltonian path within it and we can recreate a Hamiltonian path in the larger cuboid by connecting endpoints from the ending point of one cuboid to the starting point of the succeeding cuboid. For cuboids that violate this condition, there will be a required notch and an admissible cuboid subdivision is possible for all but one of the cuboid subdivisions, recursively limiting the notch violation to a single point.

3.3. 2D Generalized Hilbert Function (Gilbert2D)

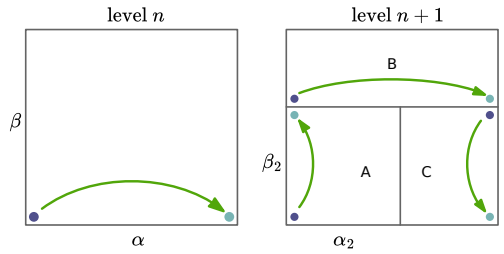

Figure 2: For the 2D Gilbert curve, the main rectangle is subdivided into three sub regions, making sure their paths connect and preferring even side lengths for the first rectangular subdivision. Partitioning the rectangle in this way is what we call a U-split.

For the 2D Gilbert curve, a U-split template is used, highlighted in Figure 2. The U-split breaks the region into three sub-blocks, labeled A, B and C. Each of the A, B and C regions have local coordinate systems that are resized, rotated and new endpoints chosen so the process can be recursively re-applied.

The sides of the subdivided blocks A, B and C are chosen to remain integral. The width-like dimension for subdivided block A is chosen to be even (β_2 in Figure 2) by dividing the height-like dimension of the original region (β in Figure 2) by two and by adding one if need be. Since the width-like length of the subdivided A and C block is even (β_2), there is a guaranteed valid Hamiltonian path in both.

Should the original width-like length be even (α in Figure 2), the C block will continue to have a valid Hamiltonian path. If the original width-like dimension is odd, a Hamiltonian path is not possible and there is a forced notch that will appear in the subdivided block C.

Figure 3 gives examples of the different parity conditions for the original area, as well as the different conditions when the A and C subdivided areas are coerced into having even parity. Figure 3 also shows when a Hamiltonian path is not possible, indicated by a red cross, and illustrates how the notch is pushed into the C block under these conditions.

If a subdivided block becomes too long in its width-like dimension, the block is divided in half and the recursion proceeds as normal. Even sides are only enforced if the length is larger than 2, with a length 2 or smaller representing the base case. The base case enumerates paths in the axis direction of length greater than one.

Since A and C are chosen to have an even width-like local axis length, no new notches are introduced and any obligatory notch is effectively "pushed" into the C block.

Algorithm 1 shows the pseudo-code for computing the 2D Gilbert curve. Note that α and β are taken to be vectors in 3D, where the third dimension can be ignored if a purely 2D curve is desired. The generalization to 3D allows the Gilbert2D function to be used unaltered when the 3D Gilbert curve needs to trace out in-plane sub-curves.

The $\delta(\cdot)$ function returns one of the six directional vectors indicating which of the major signed axis aligned directions the input vector points to ($[\pm1, 0, 0], [0, \pm1, 0], [0, 0, \pm1]$). For completeness, the function is defined in Procedure 3.

Algorithm 1 assumes standard Euclidean two norm ($|v| = \sqrt{v_0^2 + v_1^2 + v_2^2}$) and abuses notation by allowing scalar to vector multiplication ($i \in \mathbb{Z}, v \in \mathbb{Z}^3, i \cdot v \to [i \cdot v_0, i \cdot v_1, i \cdot v_2]$). where the context is clear.

Example outputs of the *Gilbert2D* function are listed in Figure **??**, showing a (4×4), (10×4) and (7×4)

Algorithm 1 2D Generalized Hilbert Function (Gilbert2D)

```
# p, α, β ∈ ℤ³
function GILBERT2D(p, α, β)

    α₂, β₂ = (α//2), (β//2)

    if (|β| ≡ 1) then
        yield p + i · δ(α) forall i ∈ |α|

    else if (|α| ≡ 1) then
        yield p + i · δ(α) 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
```

3.4. 3D Generalized Hilbert Function (Gilbert3D)

The concept of a U-split is extended into 3D by constructing a J-split. An exploded view of a J-split is shown in Figure 4.

Side lengths of the subdivided cuboids are chosen to try and not preclude a Hamiltonian path. If a path starts at the local $[0, 0, 0]$, a Hamiltonian path that ends at $[(w-1), 0, 0]$ is only possible when w is even or when all lengths are odd, including w.

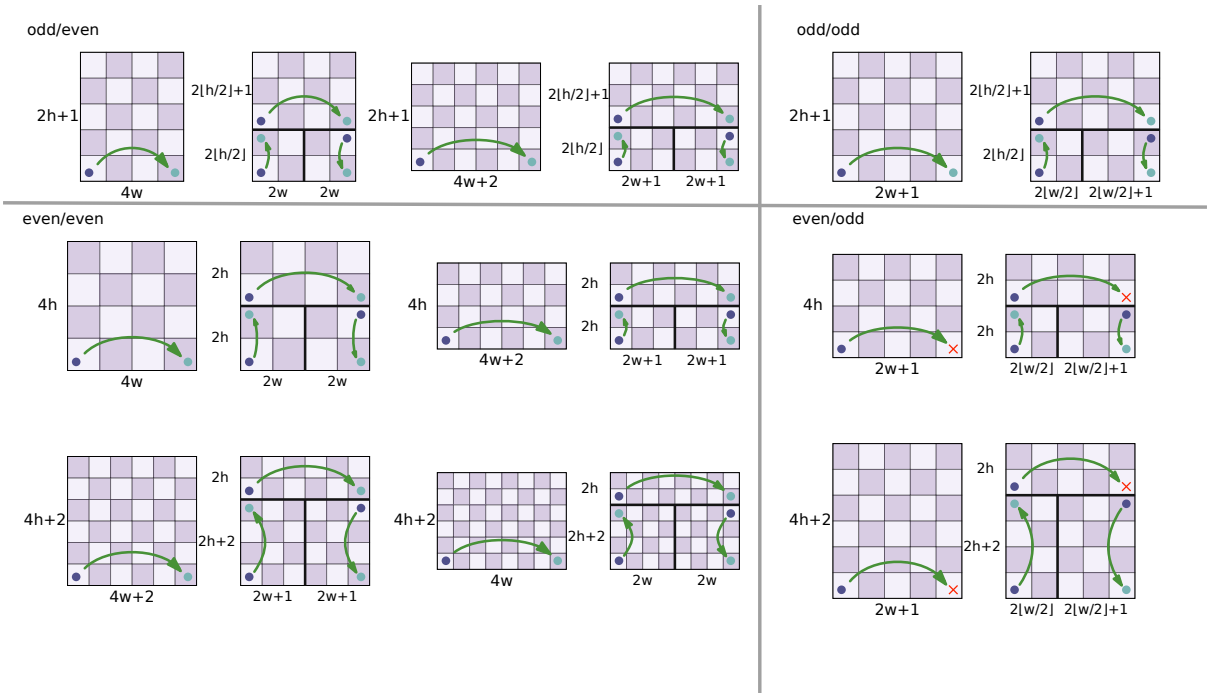

Figure 3: Enumeration of the subdivision template depending on different parities of α and β dimensions.

Algorithm 2 3D Generalized Hilbert Function (Gilbert3D)

```
# p, α, β, γ ∈ ℤ^3
function GILBERT3D(p, α, β, γ)

    # Parity of dimensions
    α0 ← (|α| mod 2)
    β0 ← (|β| mod 2)
    γ0 ← (|γ| mod 2)

    # Base cases
    if ((|α| ≡ 2) and (|β| ≡ 2) and (|γ| ≡ 2))
        return Hilbert3D(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)

    # Eccentric cases
    if (3|α| > 5|β|) and (3|α| > 5|γ|))
        return S0(p,α,β,γ)
    if (2|β| > 3|γ|) or (2|β| > 3|α|))
        return S2(p,α,β,γ)
    if (2|γ| > 3|β|)
        return S1(p,α,β,γ)

    # Bulk recursion
    return J0(p,α,β,γ) if (γ0 ≡ 0)
    return J1(p,α,β,γ) if (α0 ≡ 0)or(β0 ≡ 0)
    return J2(p,α,β,γ)
end function
```

4. Experiments and Results

5. Conclusion

References

A. Appendix

A.1. Defect

Call the *defect* a function $\lambda_d : \mathbb{N}^d \mapsto \mathbb{N}$:

$$\lambda_2(w,h) = \frac{w \cdot h}{\min(w,h)^2}$$
$$\lambda_3(w,h,d) = \frac{w \cdot h \cdot d}{\min(w,h,d)^3}$$

If there is a disjoint subdivision of a volume V_0 to $V_1 = (V_{0,0}, V_{0,1}, \ldots, V_{0,m-1})$, $V_0 = \cup_k V_{0,k}$, define the *average defect* of the subdivided volume to be:

$$\lambda_s(V_1) = \sum_k \frac{\text{Vol}(V_{0,k})}{\text{Vol}(V_0)} \cdot \lambda(V_{0,k})$$

This weights the defect of each subdivided cuboid by its proportional volume.

The defect gives us a coarse idea of how lopsided or *eccentric* a cuboid region is. If the defect is too high, we might want to split the larger sides while keeping the smaller sides the same size.

A.2. Eccentric Split Threshold

Calculations in this section will justify what threshold value to pick of when to choose an eccentric split over a J-split. Our concern is to find a simple ratio of when each of w, h or d are considered "small enough" or "large enough", relative to the other dimensions, to split on.

An enumeration of what conditions lead to an eccentric split are as follows:

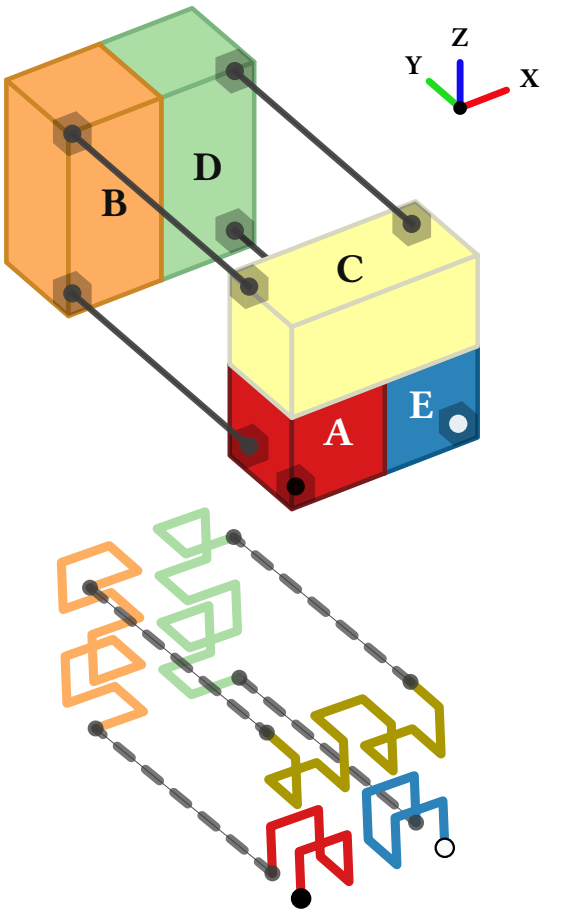

Figure 4: The J_0-split subdivision, representing the main subdivision of the bulk recursion for the 3D Gilbert curve case.

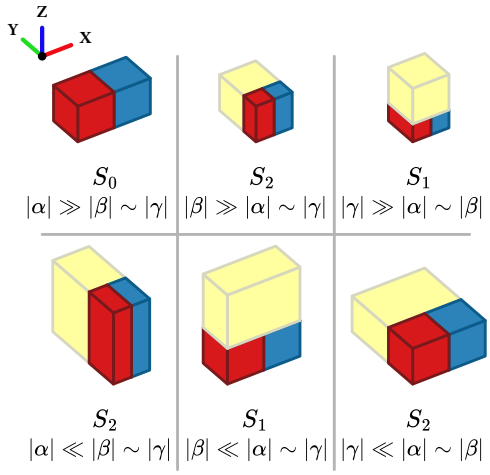

Figure 5: Eccentric cases for recursion. When the cuboid is too far from being cube-like, an S-split shape template is used to try and subdivide the cuboids into elements that are more cube like. Here α, β, γ are the width-like, height-like and depth-like dimensions. See the text or the algorithm description for the constant ratios used to determine what makes the cuboid lengths eccentric.

$$
\begin{aligned}
w >> h \sim d (1)\\
h >> w \sim d (2)\\
d >> w \sim h (3)\\
h << w \sim d (4)\\
w << h \sim d (5)\\
d << w \sim h (6)
\end{aligned}
$$

A representation of the eccentric splits are enumerated in figure **??**. The eccentric split differs from the J-split as it's only splitting in one or two dimensions, not the full three that the J-split would be doing.

For each of the six cases, we want to know what relative difference in sizes should be used to determine when an eccentric split is used and how to subdivide the cuboids so as to make progress.

Specifically, we want to find the ratio, σ or η, of when one dimension is proportionally larger or smaller, respectively, than the others and the ratio, ρ, of where to choose the split point of subdivision. For simplicity, we might want to subdivide at the half way point ($\rho = \frac{1}{2}$) but as we will see, this might give lopsided sub-cuboid regions and using a better split point is desirable.

In what follows, our goal is to choose a subdivision that will reduce the average defect. We assume that the start and end of the path lie in the w dimension with the local start point at $(0, 0, 0)$ and endpoint at $(w - 1, 0, 0)$.

Because we want to avoid adding unnecessary notches, we are forced split in the w axis. We commit to always splitting the w axis in two. If we want to subdivide into three cuboid regions, we also need to pick the split point in the other dimension another dimension

A.3. $w >> h \sim d$

Take $w = \sigma s$ and $h = d = s$. We commit to splitting the width dimension in half, subdividing the original volume into two equal volumes.

The defect of the original volume is $\lambda(w, h, d) = \sigma$.

If $w > 2h = 2s$, then $\min(\frac{w}{2}, h, d) = s$ and we have an average defect $\lambda_s(V_1) = \frac{\sigma}{2}$. Intuitively, this is validation that if the width is more than twice the length of the depth and height, we make progress if we split the width in half and recursively process each sub-cuboid.

If $w < 2h$ but $\min(\frac{w}{2}, h, d) = \frac{w}{2}$, the average defect $\lambda_s(V_1) = 2(\frac{1}{2})\frac{\frac{\sigma}{2} \cdot s^3}{(\frac{\sigma}{2} s)^3} = \frac{4}{\sigma^2}$. We want to reduce the average defect relative to the original defect, so:

$$
\begin{aligned}
& \lambda_s(V_1) < \lambda(V_0)\\
\rightarrow \quad & \tfrac{4}{\sigma^2} < \sigma\\
\rightarrow \quad & \sigma > 4^{\frac{1}{3}}\\
\rightarrow \quad & \tfrac{5}{3} > \sigma > 4^{\frac{1}{3}} = 1.58740\ldots
\end{aligned}
$$

We've chosen the constant $\frac{5}{3}$ as a simple fraction above $4^{\frac{1}{3}}$ to know when to split w.

For this case, $\sigma = \frac{5}{3}$ and $\rho = \frac{1}{2}$. That is, we split the w axis by half when the width axis exceed $\frac{5}{3}$ of both the depth, d, and height, h, dimension.

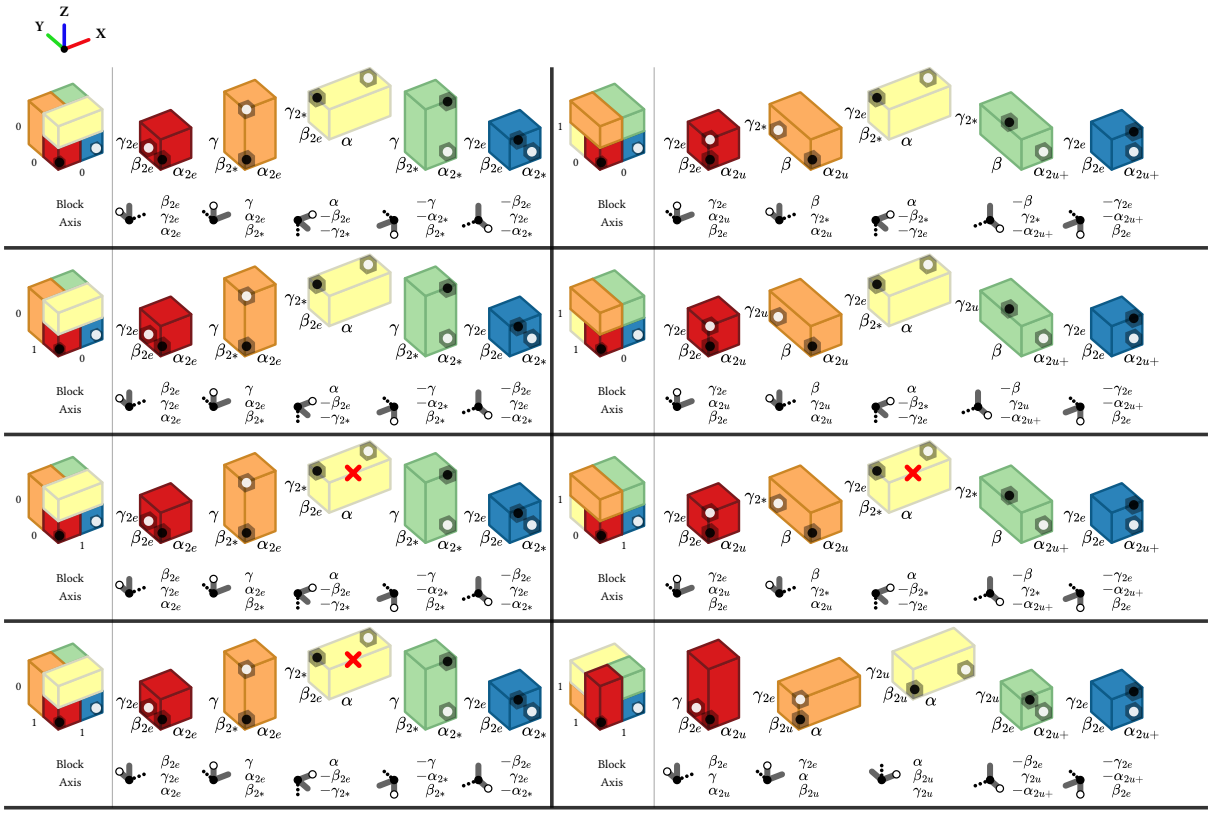

Figure 6: Bulk recursion J-split atlas for the 3D Gilbert algorithm

Procedure 3 $\delta(\cdot)$ directional vector function

```
# integral sign function
function SGN(w ∈ ℤ)
    if (w < 0) return −1
    if (w > 0) return 1
    return 0
end function

# directional vector
function δ(v ∈ ℤ³)
    return [sgn(v₀), sgn(v₁), sgn(v₂)]
end function
```

A.4. $h >> w \sim d$

B. Auxiliary Algorithms and Procedures

B.1. Helper Functions

B.2. Gilbert3D S-Split Functions

B.3. Gilbert3D J-Split Functions

Procedure 4 S_0-Split function (eccentric split)

```
# split halfway on α
function S₀(p, α, β, γ)
    α₂ ← (α//2)
    if (|α| > 2) and ((|α₂| mod 2) ≡ 1) then
        α₂ ← α₂ + δ(α)
    end if

    yield Gilbert3D(p,
                    α₂, β, γ )

    yield Gilbert3D(p + α₂,
                    (α − α₂), β, γ )
end function
```

Procedure 5 S_1-Split functions (eccentric split)

 # split $\frac{1}{3}$ on γ and halfway on α
function $S_1(p, \alpha, \beta, \gamma)$
$\alpha_2, \gamma_3 \leftarrow (\alpha//2), (\gamma//3)$
if $(|\alpha| > 2)$ and $((|\alpha_2| \bmod 2) \equiv 1)$ **then**
$\alpha_2 \leftarrow \alpha_2 + \delta(\alpha)$
end if
if $(|\gamma| > 2)$ and $((|\gamma_3| \bmod 2) \equiv 1)$ **then**
$\gamma_3 \leftarrow \gamma_3 + \delta(\gamma)$
end if

yield Gilbert3D(p, $\gamma_3, \alpha_2, \beta$)

yield Gilbert3D($p + \gamma_3$, $\alpha, \beta, (\gamma - \gamma_3)$)

yield Gilbert3D($p + \alpha - \delta(\alpha) + \gamma_3 - \delta(\gamma)$, $\gamma_3, (\alpha - \alpha_2), \beta$)
end function

Procedure 6 S_2-Split function (eccentric split)

 # split $\frac{1}{3}$ on β and halfway on α
function $S_2(p, \alpha, \beta, \gamma)$
$\alpha_2, \beta_3 \leftarrow (\alpha//2), (\beta//3)$
if $(|\alpha| > 2)$ and $((|\alpha_2| \bmod 2) \equiv 1)$ **then**
$\alpha_2 \leftarrow \alpha_2 + \delta(\alpha)$
end if
if $(|\beta| > 2)$ and $((|\beta_3| \bmod 2) \equiv 1)$ **then**
$\beta_3 \leftarrow \beta_3 + \delta(\beta)$
end if

yield Gilbert3D(p, $\beta_3, \gamma, \alpha_2$)

yield Gilbert3D($p + \beta_3$, $\alpha, (\beta - \beta_3), \gamma$)

yield Gilbert3D($p + \alpha - \delta(\alpha) + \beta_3 - \delta(\beta)$, $-\beta_3, \gamma, -\alpha$)
end function

Procedure 7 J_0-Split function

 # $|\gamma|$ even
function $J_0(p, \alpha, \beta, \gamma)$

$\alpha_2, \beta_2, \gamma_2 \leftarrow (\alpha//2), (\beta//2), (\gamma//2)$

 # prefer initial block even
$\alpha_2 = \alpha_2 + \delta(\alpha)$ **if** $(|\alpha_2| > 2)$and$(|\alpha_2| \bmod 2 \equiv 1)$
$\beta_2 = \beta_2 + \delta(\beta)$ **if** $(|\beta_2| > 2)$and$(|\beta_2| \bmod 2 \equiv 1)$
$\gamma_2 = \gamma_2 + \delta(\gamma)$ **if** $(|\gamma_2| > 2)$and$(|\gamma_2| \bmod 2 \equiv 1)$

yield Gilbert3D(p, $\beta_2, \gamma_2, \alpha_2$)

yield Gilbert3D($p + \beta_2$, $\gamma, \alpha_2, \beta - \beta_2$)

yield Gilbert3D($p + \beta_2 - \delta(\beta) + \gamma - \delta(\gamma)$, $\alpha, -\beta_2, -(\gamma - \gamma_2)$)

yield Gilbert3D($p + \alpha - \delta(\alpha) + \beta_2 + \gamma - \delta(\gamma)$, $-\gamma, -(\alpha - \alpha_2), (\beta - \beta_2)$)

yield Gilbert3D($p + \alpha - \delta(\alpha) + \beta_2 - \delta(\beta)$, $-\beta_2, \gamma_2, -(\alpha - \alpha_2)$)
end function

Procedure 8 J_1-Split function

 # $|\gamma|$ odd, one of $|\alpha|$ or $|\beta|$ even
function $J_1(p, \alpha, \beta, \gamma)$

$\alpha_2, \beta_2, \gamma_2 \leftarrow (\alpha//2), (\beta//2), (\gamma//2)$

 # prefer β_2, γ_2 even but force α_2 odd
$\alpha_2 = \alpha_2 + \delta(\alpha)$ **if** $(|\alpha_2| > 2)$and$(|\alpha_2| \bmod 2 \equiv 0)$
$\beta_2 = \beta_2 + \delta(\beta)$ **if** $(|\beta_2| > 2)$and$(|\beta_2| \bmod 2 \equiv 1)$
$\gamma_2 = \gamma_2 + \delta(\gamma)$ **if** $(|\gamma_2| > 2)$and$(|\gamma_2| \bmod 2 \equiv 1)$

yield Gilbert3D(p, $\gamma_2, \alpha_2, \beta_2$)

yield Gilbert3D($p + \gamma_2$, $\beta, \gamma - \gamma_2, \alpha_2$)

yield Gilbert3D($p + \gamma_2 - \delta(\gamma) + \beta - \delta(\beta)$, $\alpha, -(\beta - \beta_2), -\gamma_2$)

yield Gilbert3D($p + \alpha - \delta(\alpha) + \beta - \delta(\beta) + \gamma_2$, $\beta, \gamma - \gamma_2, -(\alpha - \alpha_2)$)

yield Gilbert3D($p + \alpha - \delta(\alpha) + \gamma_2 - \delta(\gamma)$, $-\gamma_2, -(\alpha - \alpha_2), \beta_2$)
end function

Procedure 9 J_2-Split function

 # $|\alpha|, |\beta|, |\gamma|$ odd
function $J_2(p, \alpha, \beta, \gamma)$

$\alpha_2, \beta_2, \gamma_2 \leftarrow (\alpha//2), (\beta//2), (\gamma//2)$

 # prefer β_2, γ_2 even but force α_2 odd
$\alpha_2 = \alpha_2 + \delta(\alpha)$ **if** $(|\alpha_2| > 2)$and$(|\alpha_2| \bmod 2 \equiv 0)$
$\beta_2 = \beta_2 + \delta(\beta)$ **if** $(|\beta_2| > 2)$and$(|\beta_2| \bmod 2 \equiv 1)$
$\gamma_2 = \gamma_2 + \delta(\gamma)$ **if** $(|\gamma_2| > 2)$and$(|\gamma_2| \bmod 2 \equiv 1)$

yield Gilbert3D(p,
$\beta_2, \gamma, \alpha_2$)

yield Gilbert3D($p + \beta_2$,
$\gamma_2, \alpha, (\beta - \beta_2)$)

yield Gilbert3D($p + \beta_2 + \gamma_2$,
$\alpha, (\beta - \beta_2), (\gamma - \gamma_2)$)

yield Gilbert3D($p + \alpha - \delta(\alpha) + \beta_2 - \delta(\beta) + \gamma_2$,
$-\beta_2, (\gamma - \gamma_2), -(\alpha - \delta(\alpha))$)

yield Gilbert3D($p + \alpha - \delta(\alpha) + \gamma_2 - \delta(\gamma)$,
$-\gamma_2, -(\alpha - \alpha_2), \beta_2$)
end function