Properties of the Zigzag Space Filling Curve

piers.barber@logicmonkey.co.uk

May 7, 2022

Abstract

Interesting properties of the Zigzag space filling curve are presented. Methods for the calculation of positions on the curve are shown, specifically a method to convert a Cartesian coordinate to a parametric distance along the curve and, the inverse of this operation.

1 Introduction

Searches for literature on the zigzag space filling curve don't yield too many results. It does not seem to have been studied to much depth in many places, presumably because there are other curves that are recursive and with simpler parametric and coordinate representations: for example, the Hilbert-Peano [2] and Morton Order (Z) [5] curves. The zigzag may be of limited use, but it does fill square regions of arbitrary width and height (non powers of two) without loss of generality.

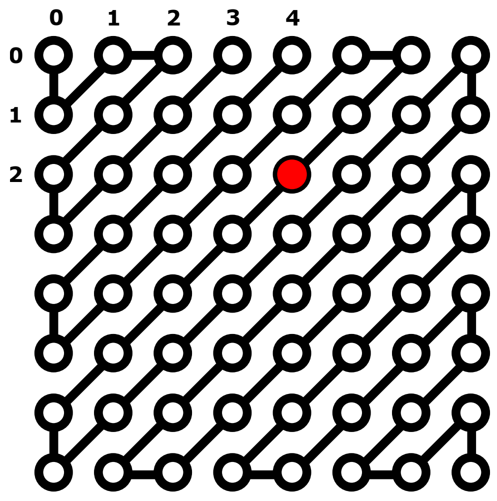

Figure 1: Example $\mathbf{8} \times \mathbf{8}$ zigzag with highlighted node S=23 ⇔ XY=(4,2)

The zigzag space filling curve of Fig. 1 is mainly used in MPEG [4] and JPEG [3] encode/decode. Implementations generally use a 64 entry look up table to convert between a linear address and an $\mathbf{8} \times \mathbf{8}$ array entry. It's a simple solution, but it hides interesting attributes of the curve that can be calculated in real time rather than looked up.

This article will show how we can:

- convert parametric distance S along the curve to Cartesian coordinates,
- convert Cartesian coordinates to parametric distance S,
- constrain the curve to a square region without the need for bounding box checks,
- implement the calculations in hardware (RTL). [1]

2 Properties

2.1 General

Fig. 2 shows points on the curve over an unbounded region. Note the following properties:

Rule 1. *The sum of the XY coordinates is constant and unique along each diagonal*

Rule 2. *Each diagonal starts with a triangular number.* [2] $\mathbf{T_n}$

Rule 3. *The sum of the XY coordinates alternates even/odd from one diagonal to the next. Adjacent diagonals run in opposite directions.*

Rule 4. *The sum of the XY coordinates along a diagonal is the triangular root* [3] *of all numbers on that diagonal.*

These rules allow us to navigate within the space using both the parametric distance along the curve and the XY position.

2.2 Bounded Regions

Fig. 3 is an example of a zigzag curve bounded to $\mathbf{5} \times \mathbf{5}$ points. Left unbounded, the curve strays outside of the required region for points to the right of the principal (longest) diagonal. Other points stray into the bounded region. We can correct this behaviour by noting that there is symmetry around the principal diagonal and reflecting each point to its right onto a corresponding correctly numbered point on its left.

[1] Python that refactors readily to an HDL for RTL synthesis. Pytherilog™?

[2] The $\mathbf{n}$th triangular number $\mathbf{T_n}$ is the sum of all natural numbers from $\mathbf{1}$ to $\mathbf{n}$

[3] For a number $\mathbf{k}$ on the diagonal, the triangular root of $\mathbf{k}$ is the natural number $\mathbf{n}$ that has the highest triangular number $\mathbf{T_n} \leq \mathbf{k}$.

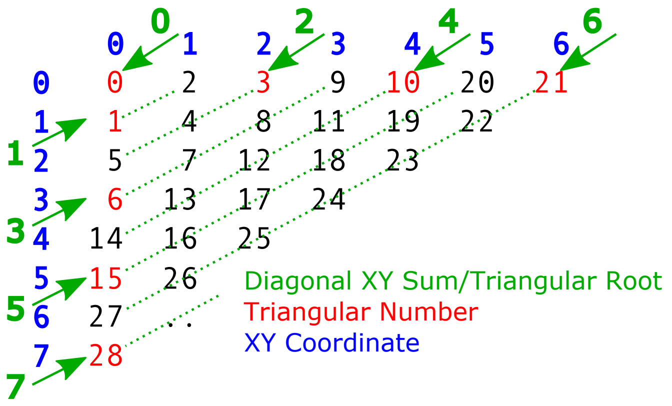

Figure 2: Unbounded zigzag

Rule 5. *For an* $\mathbf{N} \times \mathbf{N}$ *bound, the principal diagonal has an XY coordinate sum (triangular root) of* $\mathbf{x} + \mathbf{y} = \mathbf{N} - \mathbf{1}$.

Rule 6. *Points at a distance greater than half way along the curve are considered to be to the right of the principal diagonal (and have a reflectively translated counter position to the left of the principal diagonal).*

Rule 7. *Bounded diagonals to the right of the principal diagonal do not in general start on a triangular number.*

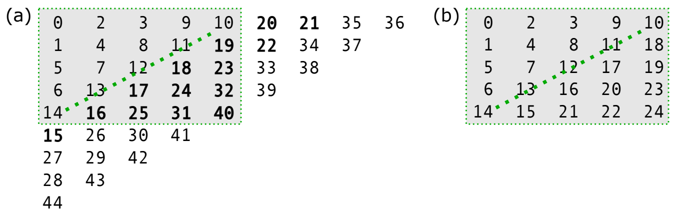

Figure 3: Bounding: (a) Unconstrained behaviour, errors in bold. (b) Constrained behaviour, reflection around principal diagonal

3 Mapping XY to distance S

We assume a bounding box of $\mathbf{N} \times \mathbf{N}$ and Cartesian coordinates $(\mathbf{x}, \mathbf{y})$ such that

$$\mathbf{0} \leq \mathbf{x} \leq \mathbf{N} - \mathbf{1}, \mathbf{0} \leq \mathbf{y} \leq \mathbf{N} - \mathbf{1}$$

To find the distance $\mathbf{S}$ along the curve for some point $\mathbf{P}$ at position $(\mathbf{x_p}, \mathbf{y_p})$, the approach is:

1. Find the triangular root of the diagonal that **P** lies on (Rule 1):

$$\mathbf{n} = \mathbf{x_p} + \mathbf{y_p}$$

2. Translate **P** to $(\mathbf{x_t}, \mathbf{y_t})$ (Rule 5):

$$(\mathbf{x_t}, \mathbf{y_t}) = \begin{cases} (\mathbf{x_p}, \mathbf{y_p}), & \text{if } \mathbf{n} \le \mathbf{N-1} \text{ no change} \\ (\mathbf{N-1-x_p}, \mathbf{N-1-y_p}), & \text{otherwise right of principal diagonal.} \end{cases}$$

3. Calculate the diagonal's triangular number starting position (Rule 2):

$$\mathbf{T_n} = \frac{\mathbf{n(n+1)}}{\mathbf{2}} \tag{1}$$

4. Offset from the diagonal's start either up and to the right with $\mathbf{x_t}$ or down and to the left with $\mathbf{y_t}$ (Rule 3):

$$\mathbf{U} = \begin{cases} \mathbf{T_n} + \mathbf{x_t}, & \text{if } \mathbf{n} \text{ is odd} \\ \mathbf{T_n} + \mathbf{y_t}, & \text{if } \mathbf{n} \text{ is even.} \end{cases}$$

5. Translate back if originally right of the principal diagonal (Rule 5):

$$\mathbf{S} = \begin{cases} \mathbf{U}, & \text{if } \mathbf{n} \le \mathbf{N-1} \text{ no change} \\ \mathbf{N^2 - 1 - U}, & \text{otherwise offset back from bottom right.} \end{cases}$$

4 Mapping distance S to XY

We assume a bounding box of $\mathbf{N} \times \mathbf{N}$ and Cartesian coordinates $(\mathbf{x}, \mathbf{y})$ such that

$$\mathbf{0} \le \mathbf{x} \le \mathbf{N-1}, \mathbf{0} \le \mathbf{y} \le \mathbf{N-1}$$

To find the position $\mathbf{P} = (\mathbf{x_p}, \mathbf{y_p})$ for a point distance **S** along the curve, the approach is:

1. Reflect **S** to a parametric point **U** to the left of the principal diagonal if **S** is more than half way along the curve's length (Rule 6):

$$\mathbf{U} = \begin{cases} \mathbf{S} & \text{if } \mathbf{S} \le \tfrac{\mathbf{1}}{\mathbf{2}}\mathbf{N^2} \\ \mathbf{N^2 - 1 - S}, & \text{otherwise.} \end{cases}$$

2. Translate **P** relative to $(\mathbf{x_t}, \mathbf{y_t})$ if **S** is more than half way along the curve's length (Rule 6):

$$(\mathbf{x_t}, \mathbf{y_t}) = \begin{cases} (\mathbf{0}, \mathbf{0}) & \text{if } \mathbf{S} \le \tfrac{\mathbf{1}}{\mathbf{2}}\mathbf{N^2} \text{ no translation required} \\ (\mathbf{N-1}, \mathbf{N-1}), & \text{otherwise.} \end{cases}$$

3. Find the triangular root of the diagonal that $\mathbf{P}$ lies on (Rule 1): [4]

$$\mathbf{n} = \left\lfloor \frac{\sqrt{\mathbf{8} \cdot \mathbf{U} + \mathbf{1}} - \mathbf{1}}{\mathbf{2}} \right\rfloor$$

4. Calculate $\mathbf{T_n}$ the diagonal's triangular number starting position (Rule 2) from equation 1

5. Calculate the offset distance from the start of diagonal:

$$\mathbf{d} = \mathbf{U} - \mathbf{T_n}$$

6. Calculate $(\mathbf{x_p}, \mathbf{y_p})$ based on the odd or even diagonal and translate back to the right of the principal diagonal as necessary.

$$(\mathbf{x_p}, \mathbf{y_p}) = \begin{cases} (\mathbf{n} - \mathbf{d}, \mathbf{d}), & \text{if } \mathbf{n} \text{ is even and } \mathbf{S} \leq \frac{\mathbf{1}}{\mathbf{2}}\mathbf{N^2} \\ (\mathbf{d}, \mathbf{n} - \mathbf{d}), & \text{if } \mathbf{n} \text{ is odd and } \mathbf{S} \leq \frac{\mathbf{1}}{\mathbf{2}}\mathbf{N^2} \\ (\mathbf{x_t} - (\mathbf{n} - \mathbf{d}), \mathbf{x_y} - \mathbf{d}), & \text{if } \mathbf{n} \text{ is even and } \mathbf{S} > \frac{\mathbf{1}}{\mathbf{2}}\mathbf{N^2} \\ (\mathbf{x_t} - \mathbf{d}, \mathbf{x_y} - (\mathbf{n} - \mathbf{d})), & \text{if } \mathbf{n} \text{ is odd and } \mathbf{S} > \frac{\mathbf{1}}{\mathbf{2}}\mathbf{N^2} \end{cases}$$

5 Conclusion

Methods have been shown that enable navigation of a space along the route of a zigzag curve. Bounding box checks can be avoided altogether and only integer calculations are required.

6 Code: XY to S

The following Python code is readily refactored to Verilog RTL. A single multiplier is required.

```
N = 5

def tri(n):
    return (n*(n+1))>>1 # Triangular number starts a diagonal

def zz_xy_to_s(x, y):
    if (x+y)%2:
        return tri(x+y) + x # for diagonals going up and right
    else:
        return tri(x+y) + y # for diagonals going down and left

def zigzag_xy_to_s(x, y):
    if (x+y<N): # left of main diagonal
        END = 0
```

[4]Solve equation 1) for $\mathbf{n}$.

```
        K = 1
        xt = x
        yt = y
    else:       # right of main diagonal: reflect
        END = N*N-1
        K = -1
        xt = N-1-x
        yt = N-1-y

    # distance s is purely a function of N, x and y
    return END + K*zz_xy_to_s(xt, yt)

for y in range(0, N):
    for x in range(0, N):
        s = zigzag_xy_to_s(x, y)
        print("{:3d} ".format(s), end='')
    print("")
```

7 Code: S to XY

The following Python code is readily refactored to Verilog RTL. Note the integer square root pipeline reimagined from an iterative algorithm [1] that performs *special long division*.

```
N = 5

import math
STAGES = int(math.log(N,2))+1

def isqrt_pipe(arem, aval, stage):
    one = 1 << 2*stage
    cmp = aval | one

    yrem = arem
    yval = aval >> 1
    if arem >= cmp:
        yrem = yrem - cmp
        yval = yval | one
    return yrem, yval

def isqrt(n):
    rem = n
    val = 0
    for ps in reversed(range(0, STAGES+1)):
        rem, val = isqrt_pipe(rem, val, ps)
    return val

def tri(n):
    return (n*(n+1))>>1 # Triangular number starts a diagonal

def tri_root(n):
    return (isqrt(n<<3|1)-1)>>1

def zz_s_to_xy(s, rpd): # distance s, right of principal diagonal
    if rpd:
        s = N*N-1 - s
        xt, yt = N-1, N-1
    tr = tri_root(s)
    t = tri(tr) # nearest triangular number less than or equal to s
    d = s-t     # distance from the triangular start of diagonal

    if tr%2:    # check if the diagonal is an odd or even one
        if rpd: # right of principal diagonal
            return xt-d, yt-tr+d
        else:
            return d, tr-d
    else:
        if rpd:
            return xt-tr+d, yt-d
        else:
            return tr-d, d

def zigzag_s_to_xy(s):
    if s<N*N/2: # left of principal diagonal
        return zz_s_to_xy(s, False)
    else:         # right of principal diagonal: reflect
        return zz_s_to_xy(s, True)

for s in range(0, N*N):
    x, y = zigzag_s_to_xy(s)
    print("{:3d} {:3d} {:3d}".format(s,x,y))
```

References

[1] Martin Guy. *Square Root by Abacus Algorithm.* URL: https://web.archive.org/web/20110608185858/http://medialab.freaknet.org/martin/src/sqrt. (accessed: 07.05.2022).

[2] *Hilbert Curve.* URL: https://en.wikipedia.org/wiki/Hilbert_curve. (accessed: 07.05.2022).

[3] *JPEG.* URL: https://en.wikipedia.org/wiki/JPEG. (accessed: 07.05.2022).

[4] *MPEG-1.* URL: https://en.wikipedia.org/wiki/MPEG-1. (accessed: 07.05.2022).

[5] *Z-Order Curve.* URL: https://en.wikipedia.org/wiki/Z-order_curve. (accessed: 07.05.2022).