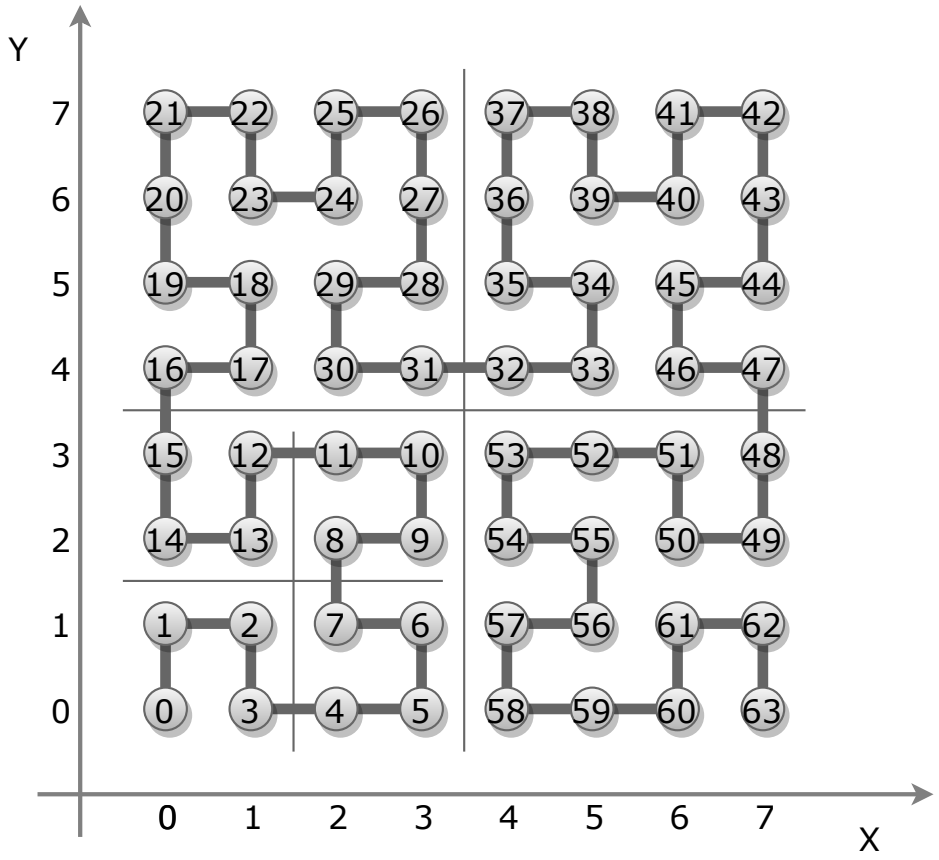
Iterative algorithm for drawing Hilbert curve

Marcin Chwedczuk | 06 AUG 2016 on Algorithms

In this post I will describe how to draw Hilbert curve iteratively. To avoid recursion we will use `hindex2xy` algorithm that translates Hilbert curve node index to Cartesian coordinates.

To index Hilbert curve nodes we assume that curve starts in the left bottom corner and ends in the right bottom corner. Indexes start at zero. Here is example numbering of `N=8` Hilbert curve:

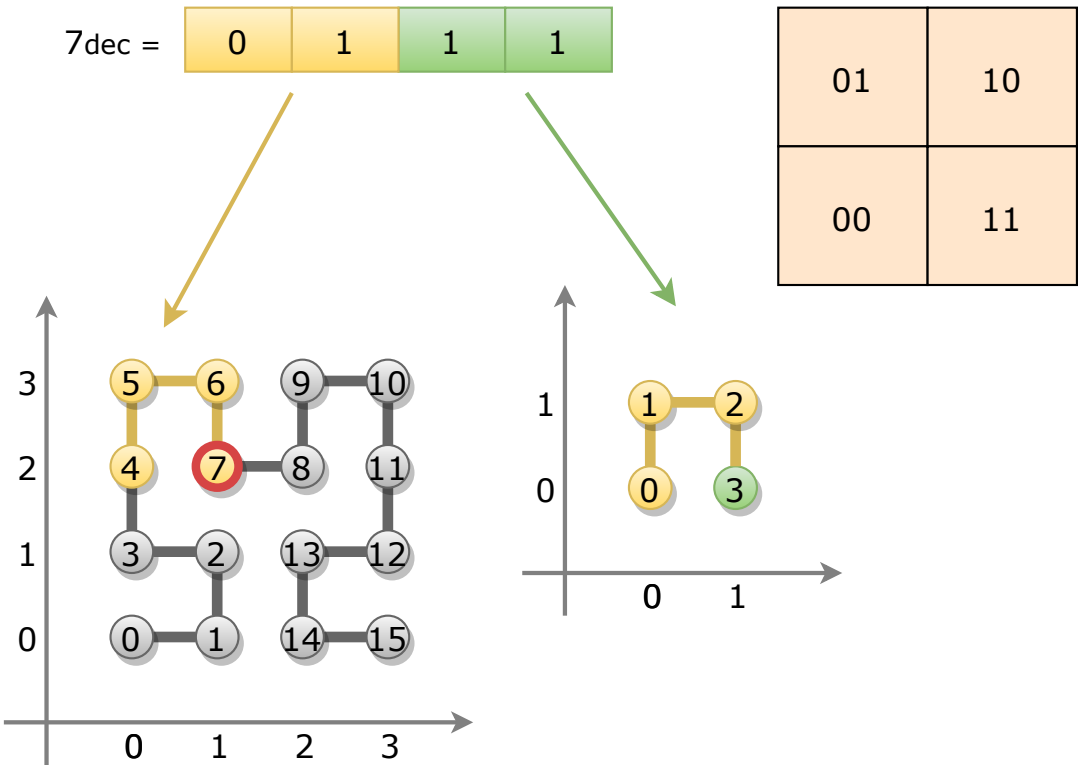


We expect that `hindex2xy(17) = (x:1, y:4)` and `hindex2xy(40) = (x:6, y:6)`.

`hindex2xy` algorithm uses bottom-up approach to compute node coordinates without using recursion. When we look at the binary representation of the node indexes we may notice that the last two bits represent node position inside `N=2` Hilbert curve. Next

two bits represent where that `N=2` Hilbert curve is located inside bigger `N=4` curve etc.

Example will show us how it works for `N=4` Hilbert curve and `index=7`:



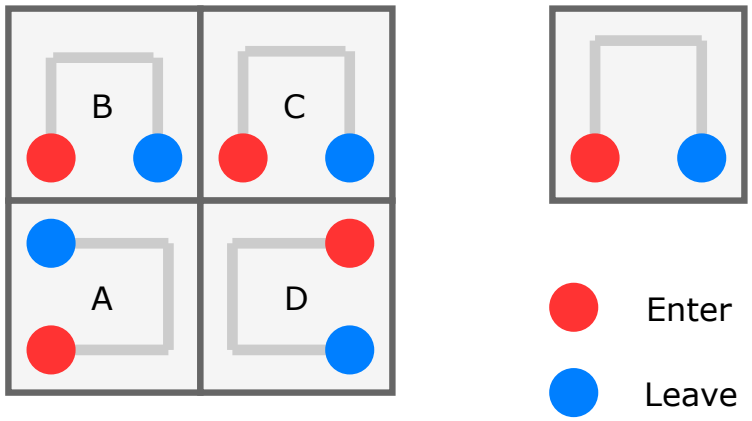
Let's start by writing index value as binary number: 7_{dec} is equal 0111_{bin}. The last two bits of 0111_{bin} are 11_{bin}, they corresponds to the bottom right node of the `N=2` Hilbert curve (marked green on the image). This node has Cartesian coordinates `(x:1, y:0)`. Now let's move to the next two bits: 01_{bin}, these bits corresponds to the left upper corner of `N=4` Hilbert curve (marked yellow on the image). To transform `N=2` into `N=4` coords we must notice that left upper corner of `N=4` curve is just translated copy of `N=2` curve. So to get `N=4` coords we must apply translation `(x:0,`

`y:2)` to `N=2` coords:

```
(x:1, y:2) = (x:1, y:0) + (x:0, y:2)
```

We end with `(x:1, y:2)` point that correctly represent node with index 7 inside `N=4` curve.

Let's assume that we want to find out coords of node in `N=2K` Hilbert curve, given that we have coords `(x,y)` of node in `N=K` curve. In general when computing coords bottom-up we have four cases:



Cases B and C are really simple since `N=2K` curve contains copies of `N=K` curve in B and C squares. In case B we should transform coords using equation:

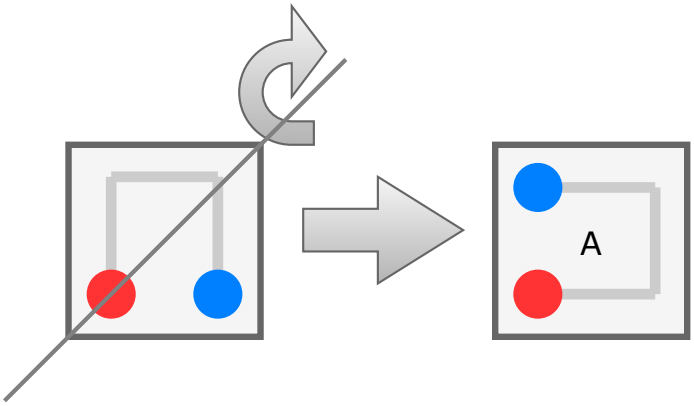
```
coords_2K = coords_K + (x:0, y:K)
```

And in case C we should use equation:

```
coords_2K = coords_K + (x:K, y:K)
```

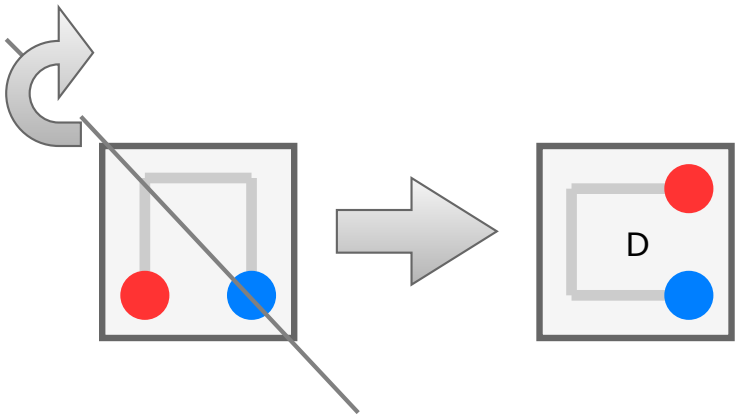
In cases A and D we must be careful when transforming coords because we must preserve order of traversal of `N=K` curve inside `N=2K` curve. I marked each curve start and end with red and blue dots so we can see better how we should perform transformation.

In case A first node of `N=K` curve should coincide with first node of `N=2K` curve and `N=K` curve should end next to the start of "case B" curve. We can achieve this by flipping coords around diagonal:




```
coords_2K.x = coords_K.y
coords_2K.y = coords_K.x
```

Case D is similar to case A, we must flip coords but around second diagonal:



In addition to that we must translate node coords to the right:

```
coords_2K.x = (K-1) - coords_K.y
coords_2K.y = (K-1) - coords_K.x

coords_2K.x = coords_2K.x + K
```

Finally let's see full algorithm code in JavaScript:

```
function last2bits(x) { return (x & 3); }

function hindex2xy(hindex, N) {
    // 1. compute position of node in N=2 curve
    var positions = [
    /* 0: */ [0, 0],
    /* 1: */ [0, 1],
    /* 2: */ [1, 1],
    /* 3: */ [1, 0]
    ];
```

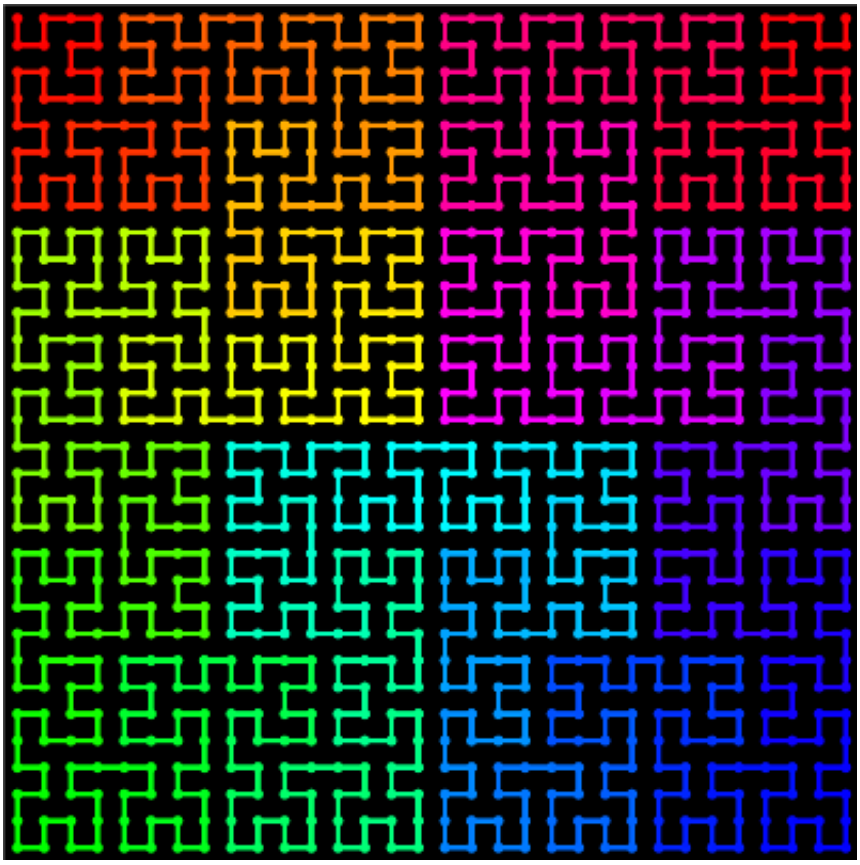
```
    var tmp = positions[last2bits(hindex)];
    hindex = (hindex >>> 2);

    // 2. iteratively compute coords
    var x = tmp[0];
    var y = tmp[1];

    for (var n = 4; n <= N; n *= 2) {
        var n2 = n / 2;

        switch (last2bits(hindex)) {
        case 0: /* case A: left-bottom */
            tmp = x; x = y; y = tmp;
            break;

        case 1: /* case B: left-upper */
            x = x;
            y = y + n2;
            break;

        case 2: /* case C: right-upper */
            x = x + n2;
            y = y + n2;
            break;

        case 3: /* case D: right-bottom */
            tmp = y;
            y = (n2-1) - x;
```

```
            x = (n2-1) - tmp;
            x = x + n2;
            break;
        }

        hindex = (hindex >>> 2);
    }
    return [x, y];
}
```

Having this algorithm we may draw Hilbert curve as follows:

```
var N = 32;

var prev = [0, 0],
    curr;

for (var i = 0; i < N*N; i += 1) {
    curr = hindex2xy(i, N);

    dot(curr);
    line(prev, curr);

    prev = curr;
}
```

And here are results:



Github

Source code: hilbert_curve

Marcin Chwedczuk

A programmer, a geek, a human

Warsaw, Poland *http://marcinchwedczuk.pl*

Share this post