Master in HPC

Problem Sheet 3 - Hilbert order

This problem is similar to the one given in PS2. Here, however, one must compute the Hilbert key for a set of points. The construction of the grid points proceeds along the lines described in PS2, but with some differences.

Let us consider a square of side length L. Write a program which computes the coordinates $xpos[i]$ and $ypos[i]$ of a set of points $N = M \times M$. The points must be arranged into the square with a uniform spacing. However, the program must compute the final list of points hierarchically, that is the program must contain a recursive function which has as input the root lattice ($N = 4 = 2 \times 2$) of points and it returns the $4N$ points which fill the four sub-squares. The order in which the sub-quadrants are filled now must be : left-bottom, left-top , right-top, right-bottom.

An important difference with respect PS2 is that here the coordinates of the points filling the sub-squares must not be a simple replica of the parent square, but their coordinates must be transformed as follows. Left-bottom : rotate $-90°$ and reverse order, left-top and right-top : identical, right-bottom rotate $+90°$ and reverse order.

The function must be recursive so that for a given recursion depth $levscan$ it proceeds until there are $4^{levscan}$ points. Finally write the final points together with the corresponding Hilbert (or $H-$) key values and make a plot of the points with a line joining them. Verify that for each point $i > 1$ it is satisfied $key(i) > key(i-1)$. Compute the $H-$keys up to the order $levhilbert = levscan$.

Assume as input $levscan = 4, L = 2^{levscan}$.

Here are given the corresponding pseudocodes.

Algorithm 1 Hilbert test

procedure POINT LATTICE ▷

Global:

Require: Int Np=4096
Require: int $rotation\_table[0:3] = (/3, 0, 0, 1/)$ ▷ set-up rotation table
Require: int $sense\_table[0:3] = (/-1, 1, 1, -1/)$ ▷ set-up direction order
Require: int $quad\_table[0:3, 0:1, 0:1]$ ▷ array of sub-qudrants
Require: real xpos[Np], ypos[Np]
Require: real quad[0:1,4] ▷ array of unit box coordinates
Require: real corner[0:1,4] ▷ array of coordinates of the unit box corners
Require: real side ▷ side length of the square
Require: int $npoints := 0$ ▷ initial number of points
Require: int $levgrid := 0$ ▷ recursion level
Require: int $Hilbert\_2D$ ▷ Hilbert key

Local:

Require: real xgrid[0:3], ygrid[0:3]
Require: int $nsub := 4$ ▷ first subdivision
Require: int $iad := 0$ ▷ address index
Require: int key_H ▷ H-key
Require: int $jstarty,\ jfiny,\ jincy$
Require: real $x\_key,\ y\_key$ ▷ input coordinates
Require: real $\Delta := 1$ ▷ point spacing unit box
Require: real $H := 0.5$ ▷ lattice shift

Begin

▷ Now construct the fundamental square. Note that the points are created in a U-reverse order, clockwise.That is : left-bottom, left-top, right-top, right-bottom; the final shape is like a $\cap$

▷ In this order we set $rotation\_table = 3, 0, 0, 1$ and $sense\_table = -1, 1, 1, -1$

    for $i \leftarrow 0, 1$ **do**
        if $i = 0$ **then**
            $jstarty := 0$ ▷ initial column index
            $jincy := 1$ ▷ increment column index
        else if $i = 1$ **then**
            $jstarty := 1$
            $jincy := -1$
        end if
        $jfiny := jstarty + jincy$ ▷ final column index
        for $j \leftarrow jstarty, jfiny, jincy$ **do**
            $iad := iad + 1$ ▷ increment address index
            $quad[0, iad] := i * \Delta + H$ ▷ set the basic points
            $quad[1, iad] := j * \Delta + H$
            $corner[0, iad] := i * \Delta$ ▷ set the corners
            $corner[1, iad] := j * \Delta$
        end for
    end for

```
    levscan := 4                                    ▷ set the final level
    side := 2^levscan                               ▷ set the side length of the square
    if 4^levscan > Np then
        print levscan, Np
        STOP
    end if
    for i ← 1, 4 do                                 ▷ map the unit square to the root square
        xgrid[i − 1] := quad[0, i] ∗ side/2
        ygrid[i − 1] := quad[1, i] ∗ side/2
    end for
    CALL makegrid_Hilbert(xgrid, ygrid, nsub)       ▷ call the recursive function
    print side
    for i ← 1, npoints do
        x_key := xpos[i]
        y_key := ypos[i]
        key_H = Hilbert2D(x_key, y_key)
        print i, xpos[i], ypos[i], key_H
    end for
end procedure
```

```
procedure MAKEGRID_HILBERT(xgrid,ygrid,n)
    input real xgrid[0 : n − 1], ygrid[0 : n − 1]
    local real xswap[0 : n − 1], yswap[0 : n − 1]
    local real xsub[0 : 4 ∗ n − 1], ysub[0 : 4 ∗ n − 1]
    local real xtemp, ytemp
    if levgrid + 1 >= levscan then
        return
    end if
    levgrid := levgrid + 1
    for isub ← 0, 3 do                              ▷ scan the four sub-quadrants
        iad = isub ∗ n
        for i ← 0, n − 1 do                         ▷ reduce to one-half
            xswap[i] := xgrid[i]/2
            yswap[i] := ygrid[i]/2
        end for
        if isub = 0 then
            for i ← 0, n − 1 do                     ▷ left-bottom: rotate +90°
                xtemp := xswap[i]
                xswap[i] := yswap[i]
                yswap[i] := side/2 − xtemp
            end for
            for i ← 0, n/2 − 1 do                   ▷ swap the point order, left-bottom first
                xtemp := xswap[i]
                ytemp := yswap[i]
                xswap[i] := xswap[n − 1 − i]
                yswap[i] := yswap[n − 1 − i]
                xswap[n − 1 − i] := xtemp
                yswap[n − 1 − i] := ytemp
            end for
        end if                                      ▷ end isub = 0
```

```
        if isub = 3 then
            for i ← 0, n − 1 do                    ▷ right-bottom: rotate −90°
                ytemp := yswap[i]
                yswap[i] := xswap[i]
                xswap[i] := side/2 − ytemp
            end for
            for i ← 0, n/2 − 1 do        ▷ swap the point order, left-bottom first
                xtemp := xswap[i]
                ytemp := yswap[i]
                xswap[i] := xswap[n − 1 − i]
                yswap[i] := yswap[n − 1 − i]
                xswap[n − 1 − i] := xtemp
                yswap[n − 1 − i] := ytemp
            end for
        end if                                                ▷ end isub = 3
        for i ← 0, n − 1 do ▷ now add the points of the sub square to the sub-quadrant
            xsub[i + iad] := xswap[i] + corner[0, isub] ∗ side/2
            ysub[i + iad] := yswap[i] + corner[1, isub] ∗ side/2
        end for
    end for                                         ▷ end quadrants loop isub

    CALL makegrid_hilbert(xsub, ysub, 4 ∗ n) ▷ repeat for the new 4*n points
    if levgrid + 1 = levscan then      ▷ end of the recursion copy to final arrays
        for m ← 1, 4 ∗ n do
            xpos[m] = xsub[m − 1]
            ypos[m] = ysub[m − 1]
        end for
        npoints = 4 ∗ n
    end if
end procedure
```

procedure HILBERT2D(x,y) ▷ compute the 2D H-key

input real x, y

local integer $rotation$, $sense$, $xbit$, $ybit$, num, k, $quadh$

▷ Here we declare the sub-quadrants orientations

▷ We use the array $quad\_table[rot, xbit, ybit]$ where $rot = rotation$; $xbit$, $ybit = 0, 1$ bit coordinates of the sub-quadrants

$quad\_table[0, 0, 0] := 0$ ▷ root order $rot = 0$ shape=∩

$quad\_table[0, 1, 0] := 3$

$quad\_table[0, 0, 1] := 1$

$quad\_table[0, 1, 1] := 2$

$quad\_table[1, 0, 0] := 1$ ▷ $rot = 1$ shape=⊂

$quad\_table[1, 1, 0] := 0$

$quad\_table[1, 0, 1] := 2$

$quad\_table[1, 1, 1] := 3$

$quad\_table[2, 0, 0] := 2$ ▷ $rot = 2$ shape=U

$quad\_table[2, 1, 0] := 1$

$quad\_table[2, 0, 1] := 3$

$quad\_table[2, 1, 1] := 0$

$quad\_table[3, 0, 0] := 3$ ▷ $rot = 3$ shape=⊃

$quad\_table[3, 1, 0] := 2$

$quad\_table[3, 0, 1] := 0$

$quad\_table[3, 1, 1] := 1$

```
    rotation := 0                                         ▷ initially no rotation
    sense := 1                                            ▷ initially sense = 1
    k := side/2
    num := 0
    while k > 0 do                      ▷ proceed until all the levels are done
        xbit := x/k                          ▷ get the most significant bit of x
        ybit := y/k                          ▷ get the most significant bit of y
        x := x − xbit ∗ k                             ▷ remove the msb from x
        y := y − ybit ∗ k                             ▷ remove the msb from y
        quadh = quad_table[rotation, xbit, ybit]            ▷ which quadrant ?
                              ▷ now evaluate the key according to the sub-square
        if sense = −1 then
            num := num + k ∗ k ∗ (3 − quadh)          ▷ travel in reverse order
        else
            num := num + k ∗ k ∗ quadh
        end if
        rotation := rotation + rotation_table[quadh]       ▷ next rotation value
        if rotation ≥ 4 then                                        ▷ module 4
            rotation := rotation − 4
        end if
        sense := sense ∗ sense_table[quadh]                ▷ next sense value
        k := k/2                                           ▷ k down one level
    end while                                                   ▷ end k > 0
    return Hilbert2D := num
end procedure
```