Parallel and Distributed Computing

Alberto Paoluzzi – Lecture 23 – Parallel algorithm

Mon 09-05-2022

Parallel algorithms: pointInPolygon

From: Paoluzzi A. A Robust, Tile-Based Algorithm for Point/Polygon Classification. Dip. di Informatica e Sistemistica, Università 'La Sapienza', Techn. Rep. 03-86, Rome, Jun 1986.

1. Problem: point in polygon classification
2. A faster tile-based technique
3. Julia sequential coding
4. Towards parallel implementation

Section 1

Problem: point in polygon classification

Problem: point in polygon classification

Classify by angle summation

How can I determine whether a 2D Point is within a Polygon?

One of longest answers on stack overflow

- Asked 13 years, 6 months ago

Classify by angle summation

How can I determine whether a 2D Point is within a Polygon?

One of longest answers on stack overflow

- Asked 13 years, 6 months ago
- Modified 2 months ago

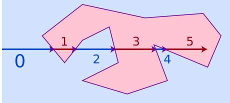
Classify by angle summation

How can I determine whether a 2D Point is within a Polygon?

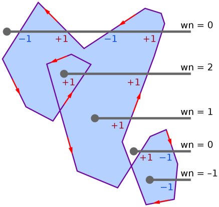
One of longest answers on stack overflow

- Asked 13 years, 6 months ago
- Modified 2 months ago
- Viewed 365k times

Classify by intersection count



The number of intersections for a ray passing from the exterior of the polygon to any point; if odd, it shows that the point lies inside the polygon. If it is even, the point lies outside the polygon; this test also works in three dimensions.



Visualization of Dan Sunday's winding number algorithm. A winding number of 0 means the point is outside the polygon; other values indicate the point is inside the polygon

Section 2

A faster tile-based technique

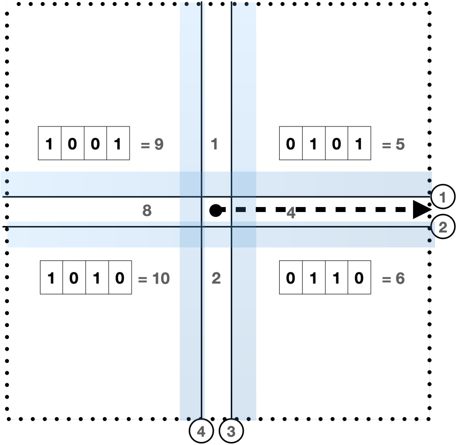
A faster tile-based technique

From: Paoluzzi A., Robust, Tile-Based Algorithm for Point/Polygon Classification. Dip. di Informatica e Sistemistica, Università 'La Sapienza', Techn. Rep. 03-86, Rome, Jun 1986.

A tile decomposition is a partition of a 2D plane by two ortogonal lines at a given point

Robust tile-based implementation

Abstract tile-based decomposition with 9 regions of different cardinality

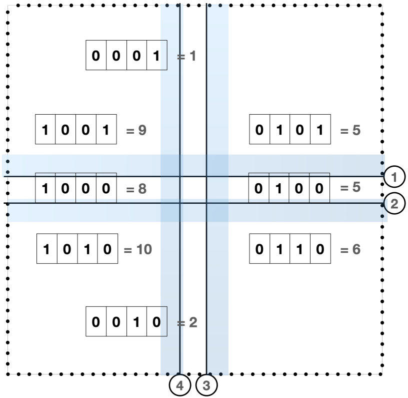


Computation of the tile code of (x, y) point

```
function setTile(box)
    # tiles = [[9,1,5],[4,0,3],[10,2,6]]
    b1,b2,b3,b4 = box
    function tileCode(point)
        x,y = point
        code = 0
        if y>b1 code=code|1 end
        if y<b2 code=code|2 end
        if x>b3 code=code|4 end
        if x<b4 code=code|8 end
        return code
    end
    return tileCode
end
```

Robust tile-based implementation: edge_code

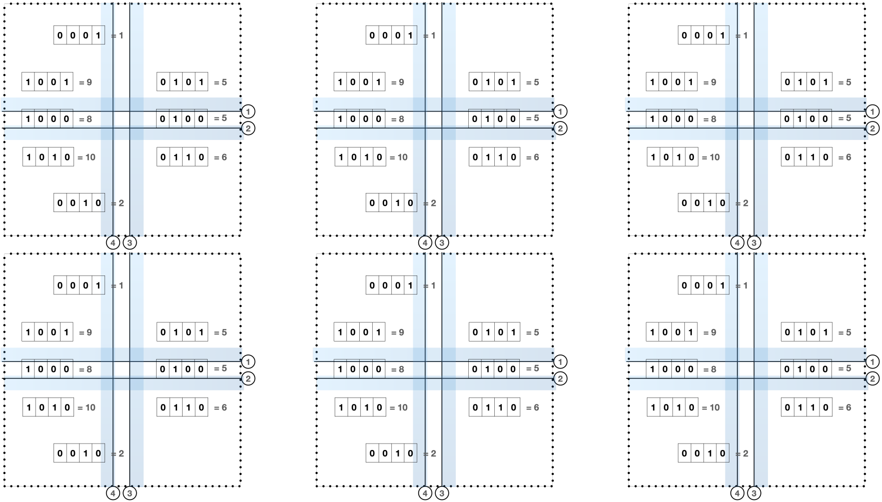
Computation of $2^4 = 16$ codes of a line segment $edge = (c_1 \vee c_2) - (c_1 \wedge c_2)$



0 0 0 0	= 0
0 0 1 1	= 3
0 1 0 0	= 4
0 1 1 1	= 7
0 1 1 1	= 7
1 0 1 1	= 11
1 1 0 0	= 12
1 1 0 1	= 13
1 1 1 0	= 14
1 1 1 1	= 15

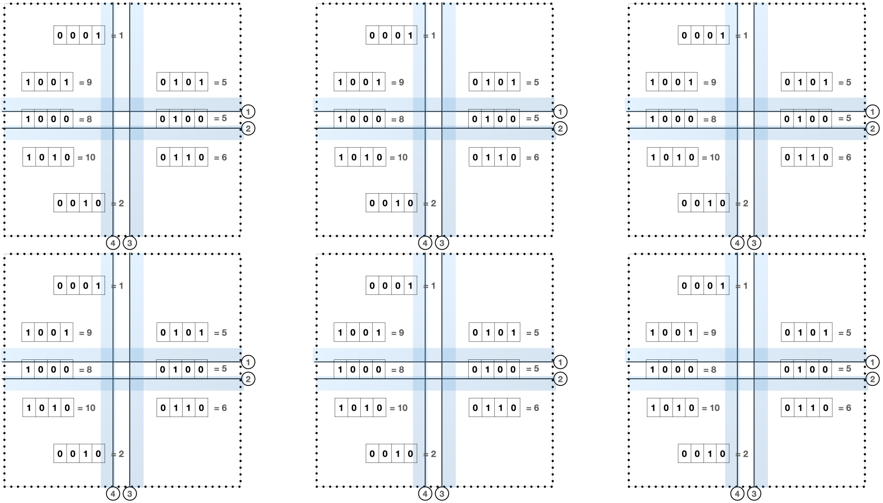
Robust tile-based implementation

Grouping line codes into equivalence classes



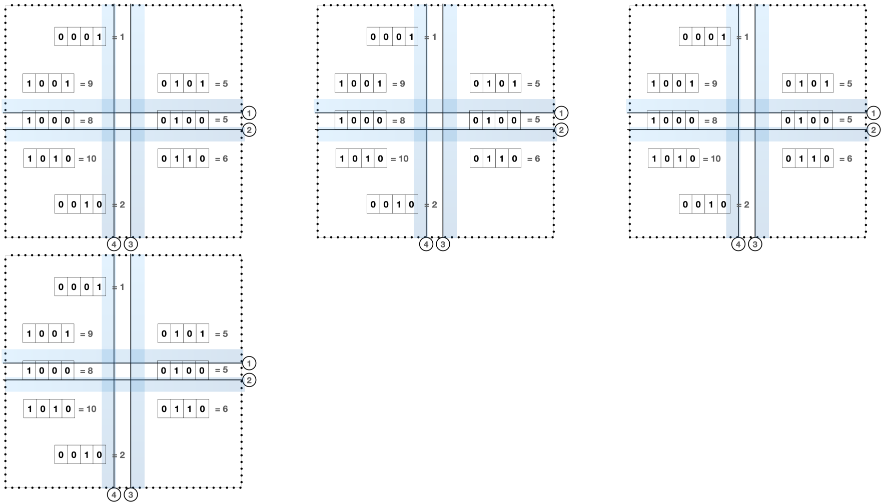
Robust tile-based implementation

Grouping line codes into equivalence classes



Robust tile-based implementation

Grouping line codes into equivalence classes



Section 3

Julia sequential coding

Julia sequential coding

Set the ‘tileCode‘ of the 2D bbox

`[b1,b2,b3,b4] == [ymax,ymin,xmax,xmin] == x,x,y,y`

including the 2D `point` of `x,y` coordinates

Depending on `point` position, `tileCode` ranges in `0:15`, and uses bit operators.

Used to set the plane tiling depending on position of the query point, in order to subsequently test the tile codes of edges of a 2D polygon, and determine if the query point is either internal, external, or on the boundary of the polygon.

Function to be parallelized . . .

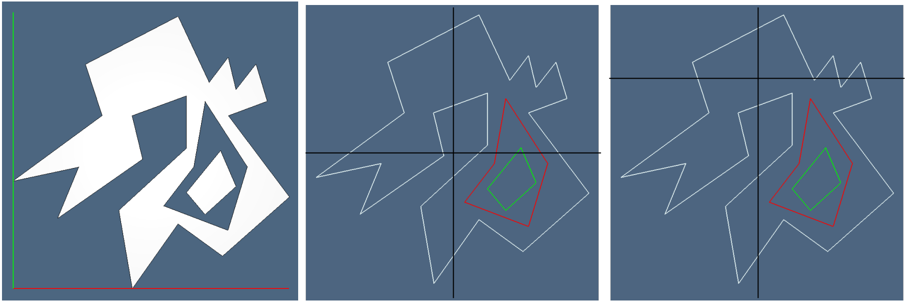
The main function

```
function pointInPolygonClassification(V,EV)
    function pointInPolygonClassification0(point)
        x,y = pnt
        xmin,xmax,ymin,ymax = x,x,y,y
        tilecode = setTile([ymax,ymin,xmax,xmin])
        count,status = 0,0

        for (k,edge) in enumerate(EV)
            p1,p2 = V[:,edge[1]],V[:,edge[2]]
            (x1,y1),(x2,y2) = p1,p2
            c1,c2 = tilecode(p1),tilecode(p2)
            c_edge, c_un, c_int = c1 xor c2, c1 or c2, c1 and c2
            ... ... ...

        if (round(count)%2)==1
            return "p_in"
        else
            return "p_out"
        end
    end
    return pointInPolygonClassification0
end
```

Auxiliary function

```julia
"""
Half-line crossing test. Utility function for `pointInPolygonClassification` functi
Update the `count` depending of the actual crossing of the tile half-line.
"""
function crossingTest(new::Int, old::Int, count::T, status::Int)::Number where T <:
    if status == 0
        status = new
        count += 0.5
    else
        if status == old
            count += 0.5
        else
            count -= 0.5
        end
        status = 0
    end
end
```

Test polygon



```
julia> Lar.pointInPolygonClassification(V,EV)((.5,.5))
"p_out"

julia> Lar.pointInPolygonClassification(V,EV)((.5,.75))
"p_in"

julia> Lar.pointInPolygonClassification(V,EV)((0.806447,0.368691))
"p_on"
```

Test application

```
(V, EV) = ([0.43145 0.596771 0.758062 1.0 0.778226 0.919353 0.
```

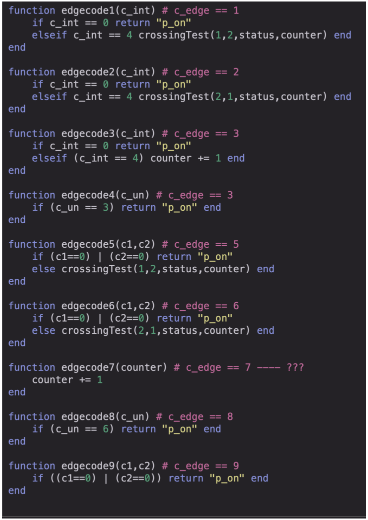
https://github.com/cvdlab/ViewerGL.jl/blob/master/examples/Polygon.jl

Section 4

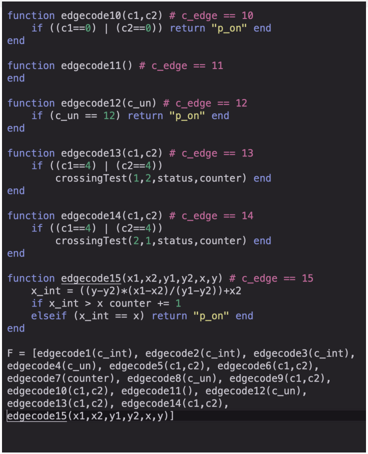
Towards parallel implementation

Towards parallel implementation

first step



```
function edgecode1(c_int) # c_edge == 1
    if c_int == 0 return "p_on"
    elseif c_int == 4 crossingTest(1,2,status,counter) end
end

function edgecode2(c_int) # c_edge == 2
    if c_int == 0 return "p_on"
    elseif c_int == 4 crossingTest(2,1,status,counter) end
end

function edgecode3(c_int) # c_edge == 3
    if c_int == 0 return "p_on"
    elseif (c_int == 4) counter += 1 end
end

function edgecode4(c_un) # c_edge == 3
    if (c_un == 3) return "p_on" end
end

function edgecode5(c1,c2) # c_edge == 5
    if (c1==0) | (c2==0) return "p_on"
    else crossingTest(1,2,status,counter) end
end

function edgecode6(c1,c2) # c_edge == 6
    if (c1==0) | (c2==0) return "p_on"
    else crossingTest(2,1,status,counter) end
end

function edgecode7(counter) # c_edge == 7 ---- ???
    counter += 1
end

function edgecode8(c_un) # c_edge == 8
    if (c_un == 6) return "p_on" end
end

function edgecode9(c1,c2) # c_edge == 9
    if ((c1==0) | (c2==0)) return "p_on" end
end
```



```
function edgecode10(c1,c2) # c_edge == 10
    if ((c1==0) | (c2==0)) return "p_on" end
end

function edgecode11() # c_edge == 11
end

function edgecode12(c_un) # c_edge == 12
    if (c_un == 12) return "p_on" end
end

function edgecode13(c1,c2) # c_edge == 13
    if ((c1==4) | (c2==4))
        crossingTest(1,2,status,counter) end
end

function edgecode14(c1,c2) # c_edge == 14
    if ((c1==4) | (c2==4))
        crossingTest(2,1,status,counter) end
end

function edgecode15(x1,x2,y1,y2,x,y) # c_edge == 15
    x_int = ((y-y2)*(x1-x2)/(y1-y2))+x2
    if x_int > x counter += 1
    elseif (x_int == x) return "p_on" end
end

F = [edgecode1(c_int), edgecode2(c_int), edgecode3(c_int),
edgecode4(c_un), edgecode5(c1,c2), edgecode6(c1,c2),
edgecode7(counter), edgecode8(c_un), edgecode9(c1,c2),
edgecode10(c1,c2), edgecode11(), edgecode12(c_un),
edgecode13(c1,c2), edgecode14(c1,c2),
edgecode15(x1,x2,y1,y2,x,y)]
```

```
julia> F = [edgecode1, edgecode2, edgecode3,
       edgecode4, edgecode5, edgecode6, edgecode7, ec
       edgecode10, edgecode11, edgecode12, edgecode13
       edgecode15]
15-element Vector{Function}:

julia> F[3]
edgecode3 (generic function with 1 method)
```