Sort points in clockwise order?

Ask Question

Given an array of x,y points, how do I sort the points of this array in clockwise order (around their overall average center point)? My goal is to pass the points to a line-creation function to end up with something looking rather "solid", as convex as possible with no lines intersecting.

For what it's worth, I'm using Lua, but any pseudocode would be appreciated. Thanks so much for any help!

Update: For reference, this is the Lua code based on Ciamej's excellent answer (ignore my "app" prefix):

```
function appSortPointsClockwise(points)
    local centerPoint = appGetCenterPointOfPoints(points)
    app.pointsCenterPoint = centerPoint
    table.sort(points, appGetIsLess)
    return points
end

function appGetIsLess(a, b)
    local center = app.pointsCenterPoint

    if a.x >= 0 and b.x < 0 then return true
    elseif a.x == 0 and b.x == 0 then return a.y > b.y
    end

    local det = (a.x - center.x) * (b.y - center.y) - (b.x - center.x) * (a.y -
center.y)
    if det < 0 then return true
    elseif det > 0 then return false
    end

    local d1 = (a.x - center.x) * (a.x - center.x) + (a.y - center.y) * (a.y -
center.y)
    local d2 = (b.x - center.x) * (b.x - center.x) + (b.y - center.y) * (b.y -
center.y)
    return d1 > d2
end

function appGetCenterPointOfPoints(points)
    local pointsSum = {x = 0, y = 0}
    for i = 1, #points do pointsSum.x = pointsSum.x + points[i].x; pointsSum.y =
pointsSum.y + points[i].y end
    return {x = pointsSum.x / #points, y = pointsSum.y / #points}
end
```

algorithm math lua geometry computational-geometry

edited Aug 15 '11 at 5:05
ciamej
4,412 1 18 33

asked Aug 8 '11 at 21:57
Philipp Lenssen
3,151 9 38 55

1 Think about compute the angle of the radial line through that point. Then sort by angle. – James K Polk Aug 8 '11 at 22:00

10 +1 Fabulous that you also shared the Lua code. – Iterator Aug 11 '11 at 1:09

In case you didn't know, lua has a builtin function `ipairs(tbl)` that iterates over the indices *and values* of tbl from 1 to #tbl. So for the sum calculation, you can do this, which most people find looks cleaner: `for _, p in ipairs(points) do pointsSum.x = pointsSum.x + p.x; pointsSum.y = pointsSum.y + p.y end` – Ponkadoodle Aug 15 '11 at 17:49

2 @Wallacoloo That's highly arguable. Also, in vanilla Lua `ipairs` is significantly slower than numeric for loop. – Alexander Gladysh Oct 5 '11 at 9:43

I had to make some small changes to get it to work for my case (just comparing two points relative to a centre). gist.github.com/personalnadir/6624172 All those comparisons to 0 in the code seem to assume that the points are distributed around the origin, as opposed to an arbitrary point. I also think that first condition will sort points below the centre point incorrectly. Thanks for the code though, it's been really helpful! – personalnadir Sep 19 '13 at 14:18

4 Answers

First, compute the center point. Then sort the points using whatever sorting algorithm you like, but use special comparison routine to determine whether one point is less than the other.

You can check whether one point (a) is to the left or to the right of the other (b) in relation to the center by this simple calculation:

```
det = (a.x - center.x) * (b.y - center.y) - (b.x - center.x) * (a.y - center.y)
```

if the result is zero, then they are on the same line from the center, if it's positive or negative, then it is

The code for the comparison function can look like this:

```
bool less(point a, point b)
{
    if (a.x - center.x >= 0 && b.x - center.x < 0)
        return true;
    if (a.x - center.x < 0 && b.x - center.x >= 0)
        return false;
    if (a.x - center.x == 0 && b.x - center.x == 0) {
        if (a.y - center.y >= 0 || b.y - center.y >= 0)
            return a.y > b.y;
        return b.y > a.y;
    }

    // compute the cross product of vectors (center -> a) x (center -> b)
    int det = (a.x - center.x) * (b.y - center.y) - (b.x - center.x) * (a.y -
center.y);
    if (det < 0)
        return true;
    if (det > 0)
        return false;

    // points a and b are on the same line from the center
    // check which point is closer to the center
    int d1 = (a.x - center.x) * (a.x - center.x) + (a.y - center.y) * (a.y -
center.y);
    int d2 = (b.x - center.x) * (b.x - center.x) + (b.y - center.y) * (b.y -
center.y);
    return d1 > d2;
}
```

This will order the points clockwise starting from the 12 o'clock. Points on the same "hour" will be ordered starting from the ones that are further from the center.

If using integer types (which are not really present in Lua) you'd have to assure that det, d1 and d2 variables are of a type that will be able to hold the result of performed calculations.

If you want to achieve something looking solid, as convex as possible, then I guess you're looking for a Convex Hull. You can compute it using the Graham Scan. In this algorithm, you also have to sort the points clockwise (or counter-clockwise) starting from a special pivot point. Then you repeat simple loop steps each time checking if you turn left or right adding new points to the convex hull, this check is based on a cross product just like in the above comparison function.

Edit:

Added one more if statement `if (a.y - center.y >= 0 || b.y - center.y >=0)` to make sure that points that have x=0 and negative y are sorted starting from the ones that are further from the center. If you don't care about the order of points on the same 'hour' you can omit this if statement and always return `a.y > b.y`.

Corrected the first if statements with adding `-center.x` and `-center.y`.

Added the second if statement `(a.x - center.x < 0 && b.x - center.x >= 0)`. It was an obvious oversight that it was missing. The if statements could be reorganized now because some checks are redundant. For example, if the first condition in the first if statement is false, then the first condition of the second if must be true. I decided, however, to leave the code as it is for the sake of simplicity. It's quite possible that the compiler will optimize the code and produce the same result anyway.

edited Jun 2 '17 at 16:19 — Arghavan — 1,118 1 5 12

answered Aug 8 '11 at 22:26 — ciamej — 4,412 1 18 33

7 +1 for a non-atan solution! – MSN Aug 8 '11 at 23:27

17 +1: No `atan()`, no square root, and even no divisions. This is a good example of computer graphics thinking. Cull off all the easy cases as soon as possible, and even in the hard cases, compute as little as possible to know the required answer. – RBerteig Aug 9 '11 at 1:29

2 +1 for Graham Scan. Good suggestion. – Iterator Aug 9 '11 at 1:47

2 if the set of points is known a priori it only takes O(n*log n) comparisons. If you want to add points in the meantime then you need to keep them in a sorted set such as a balanced binary search tree. In such a case adding a new point requires O(log n) comparisons and it's exactly the same for the solution involving polar coordinates. – ciamej Aug 9 '11 at 1:49

1 Is this missing the case: if (a.x - center.x < 0 && b.x - center.x >= 0) return false; – Tom Martin Mar 3 '14 at 23:38

What you're asking for is a system known as polar coordinates. Conversion from Cartesian to polar

answered Aug 8 '11 at 22:31

Iterator
13.7k 8 51 103

2 This will work, but it will also have the defect of doing more computation than needed to answer the ordering question. In practice, you don't actually care about either the actual angles or radial distances, just their relative order. ciamej's solution is better because it avoids divisions, square roots, and trig. – RBerteig Aug 9 '11 at 1:28

I'm not sure what your criterion is for "better". For instance, comparing all points to each other is kind of a waste of computation. Trig isn't something that scares adults, is it? – Iterator Aug 9 '11 at 1:42

2 It's not that trig is scary. The issue is that trig is expensive to compute, and wasn't needed to determine the relative order of the angles. Similarly, you don't need to take the square roots to put the radii in order. A full conversion from Cartesian to polar coordinates will do both an arc-tangent and a square root. Hence your answer is correct, but in the context of computer graphics or computational geometry it is likely to not be the *best way* to do it. – RBerteig Aug 9 '11 at 1:51

Got it. However, the OP didn't post as comp-geo, that was a tag by someone else. Still, it looks like the other solution is polynomial in the # of points, or am I mistaken? If so, that burns more cycles than trig. – Iterator Aug 9 '11 at 1:53

I hadn't actually noticed the comp-geo tag, I just assumed that the only rational applications for the question had to be one or the other. After all, the performance question becomes moot if there are only a few points, and/or the operation will be done rarely enough. At that point, knowing how to do it at all becomes important and that is why I agree your answer is correct. It explains how to compute the notion of a "clockwise order" in terms that can be explained to just about anybody. – RBerteig Aug 9 '11 at 2:04

An interesting alternative approach to your problem would be to find the approximate minimum to the Traveling Salesman Problem (TSP), ie. the shortest route linking all your points. If your points form a convex shape, it should be the right solution, otherwise, it should still look good (a "solid" shape can be defined as one that has a low perimeter/area ratio, which is what we are optimizing here).

You can use any implementation of an optimizer for the TSP, of which I am pretty sure you can find a ton in your language of choice.

edited Aug 10 '11 at 15:34

answered Aug 10 '11 at 15:01

static_rtti
18.2k 32 100 154

Yikes. "Interesting" is an understatement. :) – Iterator Aug 10 '11 at 20:02

@Iterator: I was quite happy with my idea, I was pretty disappointed to get downvoted for it :-/ Do you think it's valid? – static_rtti Aug 11 '11 at 6:45

1 I was suggesting to use one of the many fast approximations, not the NP-complete original algorithm, of course. – static_rtti Aug 11 '11 at 12:13

5 I appreciate the additional angle! To have several valid, if very different answers, might be of great help if someone in the future happens to stumble on this thread looking to brainstorm options. – Philipp Lenssen Aug 12 '11 at 9:14

1 Note that my approach is probably slower, but more correct in complex cases: imagine the case where the points for an "8", for example. Polar coordinates aren't going to help you in that case, and the result you will obtain will heavily depend on the center you chose. The TSP solution is independent of any "heuristic" parameters. – static_rtti Aug 12 '11 at 9:40

Another version (return true if a comes before b in counterclockwise direction):

```
    bool lessCcw(const Vector2D &center, const Vector2D &a, const Vector2D &b)
const
    {
        // Computes the quadrant for a and b (0-3):
        //     ^
        //   1 | 0
        //  ---+-->
        //   2 | 3

        const int dax = ((a.x() - center.x()) > 0) ? 1 : 0;
        const int day = ((a.y() - center.y()) > 0) ? 1 : 0;
        const int qa = (1 - dax) + (1 - day) + ((dax & (1 - day)) << 1);

        /* The previous computes the following:

           const int qa =
           (  (a.x() > center.x())
            ? ((a.y() > center.y())
                ? 0 : 3)
            : ((a.y() > center.y())
                ? 1 : 2)); */

        const int dbx = ((b.x() - center.x()) > 0) ? 1 : 0;
        const int dby = ((b.y() - center.y()) > 0) ? 1 : 0;
        const int qb = (1 - dbx) + (1 - dby) + ((dbx & (1 - dby)) << 1);

        if (qa == qb)
```

```
                    return (b.x() - center.x()) * (a.y() - center.y()) < (b.y() -
center.y()) * (a.x() - center.x());
                default:
                    return (center.x() - b.x()) * (a.y() - center.y()) > (b.y() -
center.y()) * (center.x() - a.x());
            }
        }
        else
        {
            return qa < qb;
        }
    }
```

This is faster, because the compiler (tested on Visual C++ 2015) doesn't generate jump to compute dax, day, dbx, dby. Here the output assembly from the compiler:

```
; 345  :        const int dax = ((a.x() - center.x()) > 0) ? 1 : 0;

    mov eax, DWORD PTR _center$[esp-4]
    xor edx, edx
    push    ebx
    push    ebp
    push    esi
    mov esi, DWORD PTR _a$[esp+8]

; 346  :        const int day = ((a.y() - center.y()) > 0) ? 1 : 0;
; 347  :        const int qa = (1 - dax) + (1 - day) + ((dax & (1 - day)) << 1);

    mov ebp, 2
    vmovsd  xmm3, QWORD PTR [eax]
    vmovsd  xmm1, QWORD PTR [eax+8]
    vxorpd  xmm2, xmm2, xmm2
    vmovsd  xmm0, QWORD PTR [esi]
    vsubsd  xmm6, xmm0, xmm3
    vmovsd  xmm0, QWORD PTR [esi+8]
    vcomisd xmm6, xmm2
    vsubsd  xmm4, xmm0, xmm1
    seta    dl
    xor ecx, ecx
    vcomisd xmm4, xmm2
    push    edi
    seta    cl
    mov edi, ebp

; 348  :         /*const int qa =
; 349  :           (  (a.x() > center.x())
; 350  :            ? ((a.y() > center.y())
; 351  :                ? 0 : 3)
; 352  :            : ((a.y() > center.y())
; 353  :                ? 1 : 2));*/
; 354  :        const int dbx = ((b.x() - center.x()) > 0) ? 1 : 0;

    xor ebx, ebx
    lea eax, DWORD PTR [ecx+ecx]
    sub edi, eax
    lea eax, DWORD PTR [edx+edx]
    and edi, eax
    sub edi, ecx
    sub edi, edx
    mov edx, DWORD PTR _b$[esp+12]
    add edi, ebp
    vmovsd  xmm0, QWORD PTR [edx]
    vsubsd  xmm7, xmm0, xmm3

; 355  :        const int dby = ((b.y() - center.y()) > 0) ? 1 : 0;

    vmovsd  xmm0, QWORD PTR [edx+8]
    vcomisd xmm7, xmm2
    vsubsd  xmm5, xmm0, xmm1
    seta    bl
    xor ecx, ecx
    vcomisd xmm5, xmm2
    seta    cl

; 356  :        const int qb = (1 - dbx) + (1 - dby) + ((dbx & (1 - dby)) << 1);

    lea eax, DWORD PTR [ecx+ecx]
    sub ebp, eax
    lea eax, DWORD PTR [ebx+ebx]
    and ebp, eax
    sub ebp, ecx
    sub ebp, ebx
    add ebp, 2

; 357  :         /*const int qb =
; 358  :            ((b.x() > center.x())
; 359  :                ? ((b.y() > center.y())
; 360  :                    ? 0 : 3)
; 361  :                : ((b.y() > center.y())
; 362  :                    ? 1 : 2));*/
; 363  :        if (qa == qb)

    cmp edi, ebp
    jne SHORT $LN57@lessCw

; 364  :        {
; 365  :            switch (qa)
```

```
; 370  :                default:
; 371  :                    return (center.x() - b.x()) * (a.y() - center.y()) >
(b.y() - center.y()) * (center.x() - a.x());

    vsubsd  xmm0, xmm3, QWORD PTR [edx]
    vmulsd  xmm1, xmm0, xmm4
    vsubsd  xmm0, xmm3, QWORD PTR [esi]
    pop edi
    vmulsd  xmm0, xmm0, xmm5
    xor eax, eax
    pop esi
    vcomisd xmm1, xmm0
    pop ebp
    seta    al
    pop ebx

; 372  :            }
; 373  :        }
; 374  :        else
; 375  :        {
; 376  :            return qa < qb;
; 377  :        }
; 378  :    }

    ret 12                  ; 0000000cH
$LN6@lessCw:
    pop edi

; 366  :            {
; 367  :                case 0:
; 368  :                case 3:
; 369  :                    return (b.x() - center.x()) * (a.y() - center.y()) <
(b.y() - center.y()) * (a.x() - center.x());

    vmulsd  xmm1, xmm7, xmm4
    vmulsd  xmm0, xmm5, xmm6
    xor eax, eax
    pop esi
    vcomisd xmm0, xmm1
    pop ebp
    seta    al
    pop ebx

; 372  :            }
; 373  :        }
; 374  :        else
; 375  :        {
; 376  :            return qa < qb;
; 377  :        }
; 378  :    }

    ret 12                  ; 0000000cH
$LN57@lessCw:
    pop edi
    pop esi
    pop ebp
    setl    al
    pop ebx
    ret 12                  ; 0000000cH
```

Enjoy.

edited Oct 8 '17 at 20:03

answered Oct 8 '17 at 19:49

AGPX

154 2 9