Sort Points on a Circle

David Eberly, Geometric Tools, Redmond WA 98052
https://www.geometrictools.com/

This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by/4.0/ or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

Created: February 25, 2023

Contents

1 Introduction 2

2 Sorting by Angle 2

3 Theoretically Correct Sorting Using Geometry 5

3.1 Case $y_0 < 0$ and $y_1 < 0$ 6

3.2 Case $y_0 > 0$ and $y_1 > 0$ 6

3.3 Case $y_0 < 0$ and $y_1 = 0$ 7

3.4 Case $y_0 < 0$ and $y_1 > 0$ 7

3.5 Case $y_0 = 0$ and $y_1 < 0$ 8

3.6 Case $y_0 > 0$ and $y_1 < 0$ 8

3.7 Case $y_0 = 0$ and $y_1 > 0$ 8

3.8 Case $y_0 > 0$ and $y_1 = 0$ 8

3.9 Case $y_0 = 0$ and $y_1 = 0$ 8

4 C++ Implementation of the Sorting Algorithm 8

1 Introduction

Given a set of points $\boldsymbol{P}_i$ for $0 \leq i < n$ and a center point $\boldsymbol{C}$ that is not in that set, sort the points in a circular manner around the center point. The sorting order can be counterclockwise (CCW) or clockwise (CW).

The algorithm involves computing vectors $\boldsymbol{V}_i = \boldsymbol{P}_i - \boldsymbol{C}$ for use by a less-than comparison function. The problem statement is then: Given a set of vectors $\boldsymbol{V}_i$ for $0 \leq i < n$, sort the points in a circular manner around the origin $\boldsymbol{O} = (0, 0)$. The sorting order can be CCW or CW.

If the vectors are unit length, the problem is further reduced to ordering the vectors on a circle centered at the origin and having radius 1. They can be sorted by an angle measured from a *reference ray* whose initial point is $\boldsymbol{O}$ and whose direction is the unit-length vector $\boldsymbol{D} = (d_0, d_1)$. Duplicate vectors can be kept or discarded as desired.

Generally, the vectors are not unit length, but they still can be sorted by an angle measured from a reference ray. It is possible that two vectors have the same angle measured from the reference ray but have different lengths. The less-than comparison function must have logic to deal with these cases.

For CCW sorting, define the vector $\boldsymbol{D}^{\perp} = (-d_1, d_0)$, which is perpendicular to $\boldsymbol{D}$ and $\{\boldsymbol{O}; \boldsymbol{D}, \boldsymbol{D}^{\perp}\}$ is a right-handed coordinate system called the *CCW reference coordinate system*. The vectors $\boldsymbol{V}_i$ can be represented in the CCW reference coordinate system by $\boldsymbol{W}_i = (x_i, y_i)$ where $\boldsymbol{V}_i = x_i\boldsymbol{D} + y_i\boldsymbol{D}^{\perp}$; that is, $x_i = \boldsymbol{D} \cdot \boldsymbol{V}_i$ and $y_i = \boldsymbol{D}^{\perp} \cdot \boldsymbol{V}_i$. The problem statement now becomes: Given a set of vectors $\boldsymbol{V}_i$ for $0 \leq i < n$, CCW-sort the points in a circular manner around $\boldsymbol{O}$ using the reference ray with direction $(1, 0)$.

For CW sorting, $\{\boldsymbol{O}; \boldsymbol{D}, -\boldsymbol{D}^{\perp}\}$ is a left-handed coordinate system which is called the *CW reference coordinate system*. The vectors $\boldsymbol{V}_i$ can be represented in the CW-reference coordinate system by $\boldsymbol{W}_i = (x_i, y_i)$ where $x_i = \boldsymbol{D} \cdot \boldsymbol{V}_i$ and $y_i = -\boldsymbol{D}^{\perp} \cdot \boldsymbol{V}_i$. The problem statement now becomes: Given a set of vectors $\boldsymbol{V}_i$ for $0 \leq i < n$, CW-sort the points in a circular manner around $\boldsymbol{O}$ using the reference ray with direction $(1, 0)$.

Use of the reference coordinate systems reduces the problem to sorting of vectors CCW about the origin with reference ray direction $(1, 0)$.

2 Sorting by Angle

The standard way to sort vectors about the origin is to use the `atan2(y,x)` function [1]. The vector input is (x, y) whose components are finite floating-point numbers. The output is an angle $\theta \in (-\pi, \pi]$, which is measured CCW from the positive x-axis, which is the reference ray direction. That is, for $y > 0$ the angle is positive and for $y < 0$ the angle is negative. When $x > 0$ and $y = 0$, the angle is 0. The function is undefined for $(x, y) = (0, 0)$. When $x < 0$ and $y = 0$, the angle is $+\pi$. For $x < 0$ and $y < -\epsilon < 0$ for a small positive ϵ, the angle is just slightly larger than $-\pi$.

Figure 1 illustrates the range of `atan2` values. The `atan2` range is open on the left because $-\pi$ is not in the range. It is closed on the right because π is in the range.

Figure 1. The range of atan2 is $(-\pi, \pi]$, visualized as a half-open/half-closed circular arc. The drawing was generated using Mathematica [2].

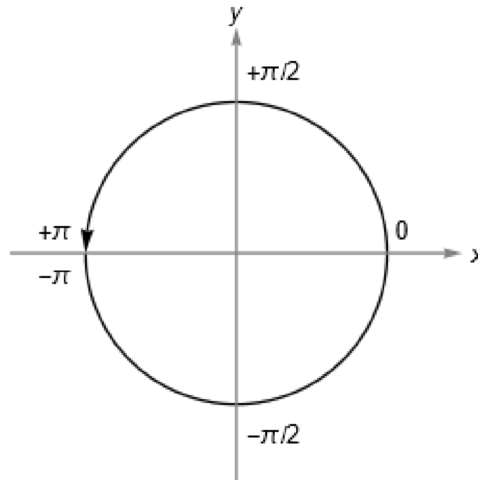


Figure 2 shows typical examples when the vectors are unit length. The reference ray and its associated quantities are red, the vectors are blue, and the polyline associated with the sorted vectors is green. The figure shows the sorted vectors $\{\boldsymbol{V}_0, \boldsymbol{V}_1, \boldsymbol{V}_2, \boldsymbol{V}_3, \boldsymbol{V}_4\}$ as the vertices of a polyline.

Figure 2. Left: The directed arc $\langle \boldsymbol{V}_0, \boldsymbol{V}_4 \rangle$ has an acute angle. Right: The directed arc $\langle \boldsymbol{V}_0, \boldsymbol{V}_4 \rangle$ has an obtuse angle. The figures were created using Mathematica [2].

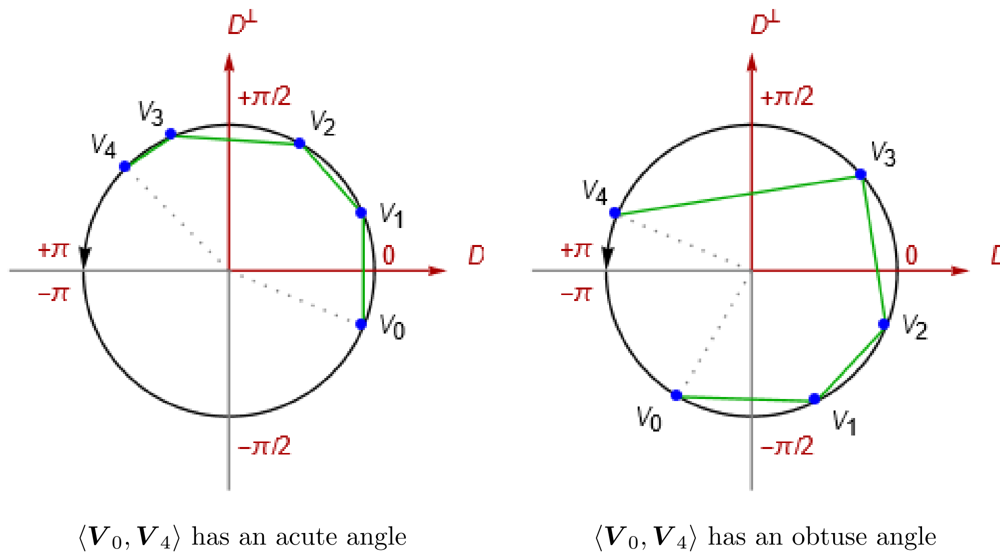
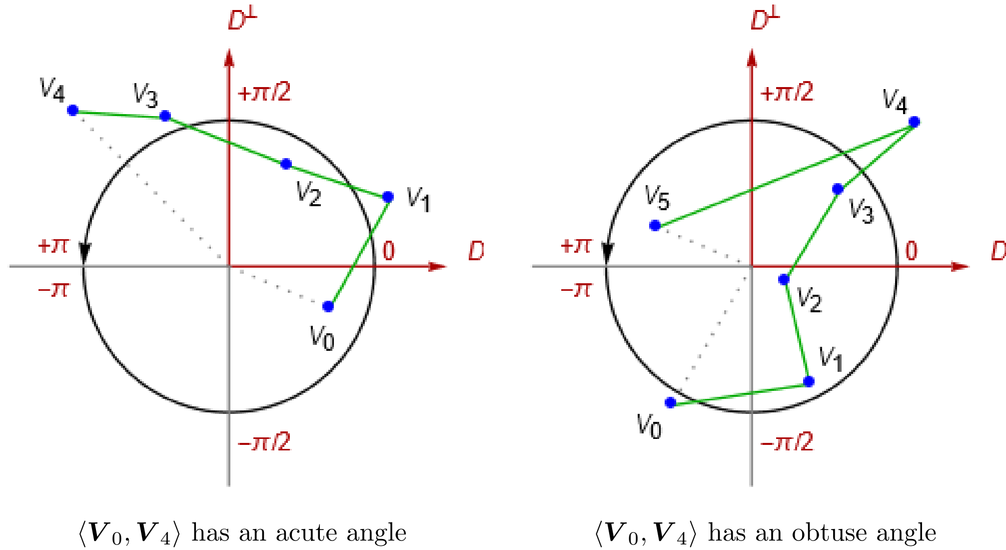


Figure 3 shows examples similar to those of Figure 2, but the vectors are not necessarily unit length.

Figure 3. Left: The directed arc $\langle \boldsymbol{V}_0, \boldsymbol{V}_4 \rangle$ has an acute angle. Right: The directed arc $\langle \boldsymbol{V}_0, \boldsymbol{V}_4 \rangle$ has an obtuse angle. Also, the vectors $\boldsymbol{V}_3$ and $\boldsymbol{V}_4$ are parallel and in the same direction, but they are not the same length. The figures were created using Mathematica [2].



$\langle \boldsymbol{V}_0, \boldsymbol{V}_4 \rangle$ has an acute angle $\quad$ $\langle \boldsymbol{V}_0, \boldsymbol{V}_4 \rangle$ has an obtuse angle

Listing 1 contains C++ code for sorting the points using a specified reference ray direction and the sorting order.

Listing 1. C++ code for sorting points P[] about center C with reference ray direction D and specified sorting order sortCCW. D need only be nonzero; it does not actually have to be unit length. Type T is a floatint-point type. The output indices[] are the sorted indices so that the sorted points are P[indices[0]], P[indices[1]], . . . , P[indices[P.size()-1]].

```
template <typename T>
struct SortObject
{
    SortObject() : W{ static_cast<T>(0), static_cast<T>(0) }, index(0) {}
    std::array<T, 2> W;
    size_t index;
};

template <typename T>
bool LessThanByAngle(SortObject<T> const& object0, SortObject<T> const& object1)
{
    T const& x0 = object0.W[0], y0 = object0.W[1];
    T const& x1 = object1.W[0], y1 = object1.W[1];
    T angle0 = std::atan2(y0, x0);
    T angle1 = std::atan2(y1, x1);
    if (angle0 < angle1)
    {
        return true;
    }
    if (angle0 > angle1)
    {
        return false;
    }
```

```
    return x0 * x0 + y0 * y0 < x1 * x1 + y1 * y1;
}

template <typename T>
void Sort(std::vector<std::array<T,2>> const& P, std::array<T,2> const& C,
    std::array<T,2> const& D, bool sortCCW, std::vector<size_t>& indices)
{
    std::array<T,2> Dperp = (sortCCW ? std::array<T,2>{ -D[1], D[0] } : std::array<T,2>{ D[1], -D[0] });
    std::vector<SortObject<T>> objects(P.size());
    for (size_t i = 0; i < P.size(); ++i)
    {
        std::array<T,2> V = P[i] - C;
        objects[i].W = { D[0] * V[0] + D[1] * V[1], Dperp[0] * V[0] + Dperp[1] * V[1] };
        objects[i].index = i;
    }
    std::sort(objects.begin(), objects.end(), LessThanByAngle<T>);
    indices.resize(P.size());
    for (size_t i = 0; i < P.size(); ++i)
    {
        indices[i] = object[i].index;
    }
}
```

3 Theoretically Correct Sorting Using Geometry

The inputs P[], C and D are finite floating-point numbers that are assumed to be exact; that is, the sorting algorithm has no knowledge of any errors that occurred during the computation of those inputs. The code in listing 1 uses floating-point arithmetic, which has rounding errors, and std::atan2, which has no closed-form representation using only arithmetic operations and thereby introduces mathematical error. Therefore, the output indices are not guaranteed to be theoretically correct because of these errors in the sorting algorithm.

Table 1 contains the logic for when $\boldsymbol{W}_0 < \boldsymbol{W}_1$ is a true statement. The cases are discussed after this table.

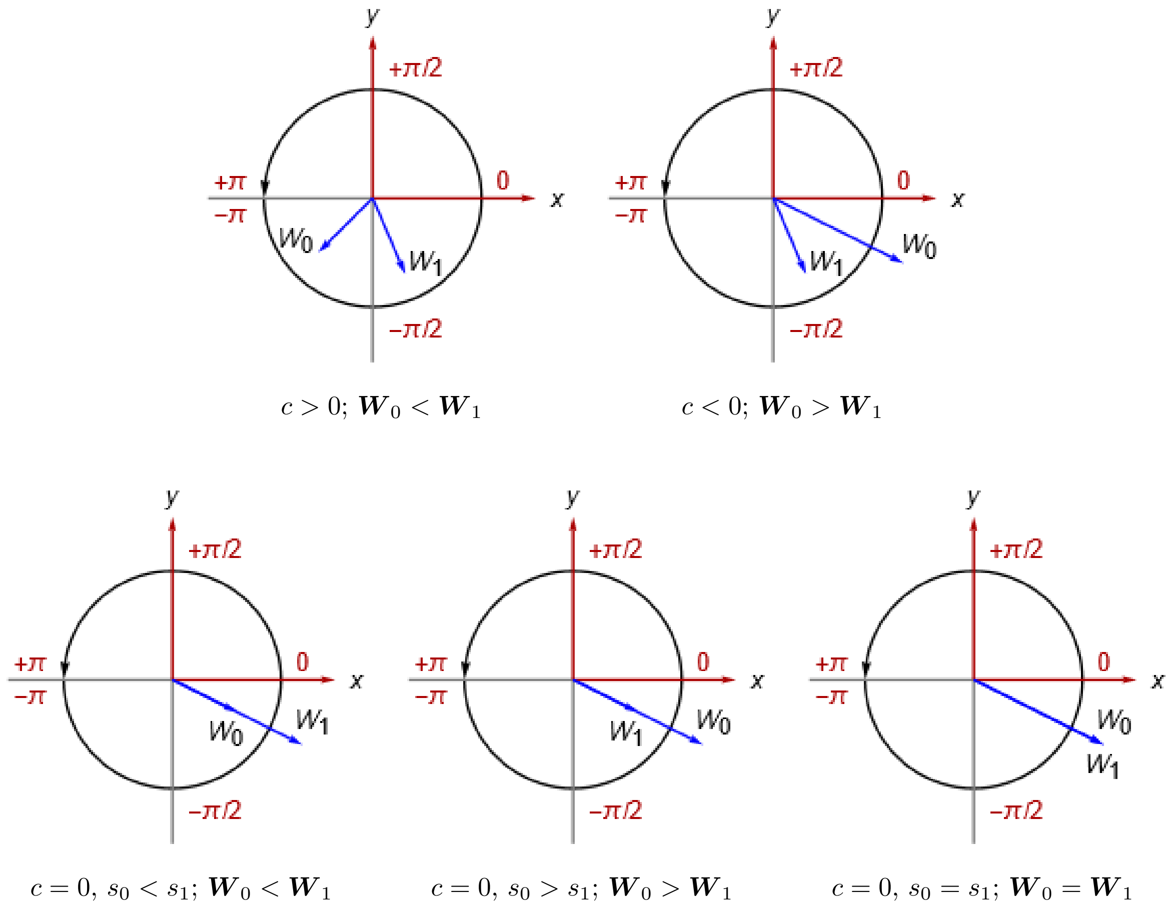
Table 1. Logic for when $\boldsymbol{W}_0 < \boldsymbol{W}_1$ is a true statement. Define $\boldsymbol{W}_i = (x_i, y_i)$ and $s_i = |\boldsymbol{W}_i|^2$ for $i \in \{0, 1\}$ and define $c = \boldsymbol{W}_0 \cdot \boldsymbol{W}_1^{\perp} = x_0 y_1 - x_1 y_0$.

	$y_1 < 0$	$y_1 > 0$	$y_1 = 0$
$y_0 < 0$	$(c > 0)$ or $(c = 0$ and $s_0 < s_1)$	true	true
$y_0 > 0$	false	$(c > 0)$ or $(c = 0$ and $s_0 < s_1)$	$x_1 < 0$
$y_0 = 0$	false	$x_0 > 0$	$(x_1 < 0$ and $x_1 < x_0)$ or $(x_0 > 0$ and $x_1 > x_0)$

3.1 Case $y_0 < 0$ and $y_1 < 0$

Figure 4 illustrates the case $y_0 < 0$ and $y_1 < 0$. Embed the 2D vectors into 3D vectors, $(\boldsymbol{W}_0, 0) = (x_0, y_0, 0)$ and $(\boldsymbol{W}_1, 0) = (x_1, y_1, 0)$. The cross product is $(\boldsymbol{W}_0, 0) \times (\boldsymbol{W}_1, 0) = (\boldsymbol{O}, c) = (0, 0, c)$. The ordering of $\boldsymbol{W}_0$ and $\boldsymbol{W}_1$ depends on whether $c > 0$, whereby $\boldsymbol{W}_1$ is a CCW rotation of $\boldsymbol{W}_0$, or $c < 0$, whereby $\boldsymbol{W}_0$ is a CCW rotation of $\boldsymbol{W}_1$. If $c = 0$, then $\boldsymbol{W}_0$ and $\boldsymbol{W}_1$ are parallel and in the same direction.

Figure 4. The case $y_0 < 0$ and $y_1 < 0$.

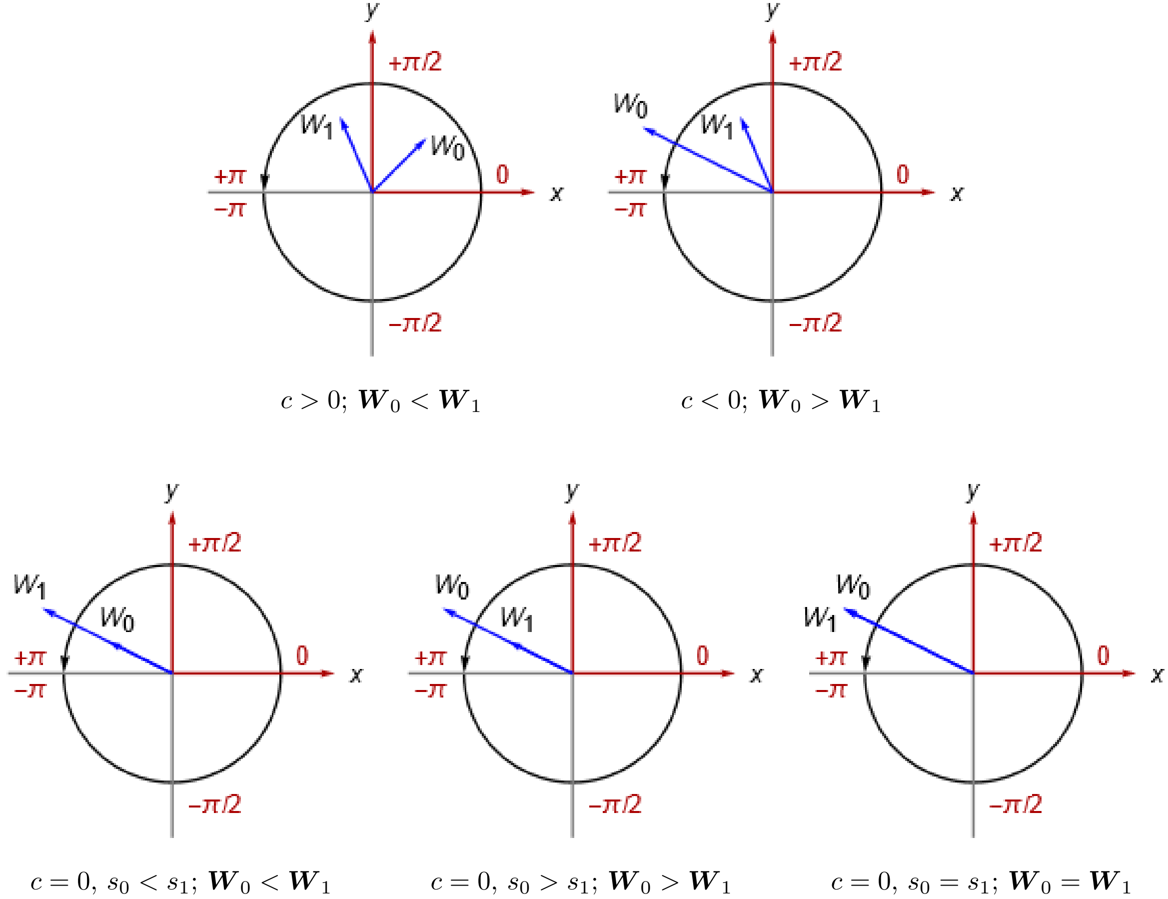


The condition for which $\boldsymbol{W}_0 < \boldsymbol{W}_1$ is a true statement is summarized by $(c > 0)$ or $(c = 0$ and $s_0 < s_1)$.

3.2 Case $y_0 > 0$ and $y_1 > 0$

Figure 5 illustrates the case $y_0 > 0$ and $y_1 > 0$. The subcases are the same as those when $y_0 < 0$ and $y_1 < 0$; see Figure 4 for a comparison.

Figure 5. The case $y_0 > 0$ and $y_1 > 0$.



The condition for which $\boldsymbol{W}_0 < \boldsymbol{W}_1$ is a true statement is summarized by $(c > 0)$ or $(c = 0$ and $s_0 < s_1)$.

3.3 Case $y_0 < 0$ and $y_1 = 0$

$\boldsymbol{W}_0$ is below the y-axis, so the angle θ_0 associated with it via `std::atan2` is negative. $\boldsymbol{W}_1$ is on the y-axis, so the angle θ_1 associated with it is either 0 when $x_1 > 0$ or π when $x_1 < 0$. In either case, $\theta_0 < 0 \leq \theta_1$ and $\boldsymbol{W}_0 < \boldsymbol{W}_1$ is a true statement.

3.4 Case $y_0 < 0$ and $y_1 > 0$

$\boldsymbol{W}_0$ is below the y-axis, so the angle θ_0 associated with it via `std::atan2` is negative. $\boldsymbol{W}_1$ is above the y-axis, so the angle θ_1 associated with it is positive. In either case, $\theta_0 < 0 < \theta_1$ and $\boldsymbol{W}_0 < \boldsymbol{W}_1$ is a true statement.

3.5 Case $y_0 = 0$ and $y_1 < 0$

$\boldsymbol{W}_0$ is on the y-axis, so the angle θ_0 associated with it via std::atan2 is 0 when $x_0 > 0$ or π when $x_0 < 0$. $\boldsymbol{W}_1$ is below the y-axis, so the angle θ_1 associated with it is negative. In either case, $\theta_1 < 0 \leq \theta_0$ and $\boldsymbol{W}_1 < \boldsymbol{W}_0$ is a false statement.

3.6 Case $y_0 > 0$ and $y_1 < 0$

$\boldsymbol{W}_0$ is above the y-axis, so the angle θ_0 associated with it via std::atan2 is positive. $\boldsymbol{W}_1$ is below the y-axis, so the angle θ_1 associated with it is negative. In either case, $\theta_1 < 0 < \theta_0$ and $\boldsymbol{W}_1 < \boldsymbol{W}_0$ in which case $\boldsymbol{W}_0 < \boldsymbol{W}_1$ is a false statement.

3.7 Case $y_0 = 0$ and $y_1 > 0$

$\boldsymbol{W}_0$ is on the y-axis, so the angle θ_0 associated with it via std::atan2 is 0 when $x_0 > 0$ or π when $x_0 < 0$. $\boldsymbol{W}_1$ is above the y-axis, so the angle θ_1 associated with it is positive. When $x_0 > 0$ we know that $\theta_0 = 0 < \theta_1$, so $\boldsymbol{W}_0 < \boldsymbol{W}_1$ is a true statement. When $x_0 < 0$ we know that $\theta_1 < \pi = \theta_0$, so $\boldsymbol{W}_0 < \boldsymbol{W}_1$ is a false statement.

3.8 Case $y_0 > 0$ and $y_1 = 0$

$\boldsymbol{W}_0$ is above the y-axis, so the angle θ_0 associated with it via std::atan2 is positive. $\boldsymbol{W}_1$ is on the y-axis, so the angle θ_1 associated with it is 0 when $x_1 > 0$ or π when $x_1 < 0$. When $x_1 > 0$ we know that $\theta_1 = 0 < \theta_0$, so $\boldsymbol{W}_0 < \boldsymbol{W}_1$ is a false statement. When $x_1 < 0$ we know that $\theta_1 = \pi > \theta_0$, so $\boldsymbol{W}_0 < \boldsymbol{W}_1$ is a true statement.

3.9 Case $y_0 = 0$ and $y_1 = 0$

Both $\boldsymbol{W}_0$ and $\boldsymbol{W}_1$ are on the y-axis. If $x_0 > 0$ and $x_1 > 0$, then $\boldsymbol{W}_0 < \boldsymbol{W}_1$ is a true statement whe $\boldsymbol{W}_0$ is closer to the origin; that is, $0 < x_0 < x_1$. If $x_0 < 0$ and $x_1 < 0$, then $\boldsymbol{W}_0 < \boldsymbol{W}_1$ is a true statement when $\boldsymbol{W}_0$ is closer to the origin; that is, $x_1 < x_0 < 0$. If $x_0 < 0$ and $x_1 > 0$, then $\theta_0 = pi$ and $\theta_1 = 0$ which implies $\boldsymbol{W}_0 < \boldsymbol{W}_1$ is a false statement. If $x_0 > 0$ and $x_1 < 0$, then $\theta_0 = 0$ and $\theta_1 = \pi$ which implies $\boldsymbol{W}_0 < \boldsymbol{W}_1$ is a true statement.

In summary, $\boldsymbol{W}_0 < \boldsymbol{W}_1$ is a true statement when $(0 < x_0 < x_1)$ or $(x_1 < x_0 < 0)$ or $(x_1 < 0 < x_0)$. An equivalent Boolean expression is $(x_1 < 0$ and $x_1 < x_0)$ or $(x_0 > 0$ and $x_1 > x_0)$.

4 C++ Implementation of the Sorting Algorithm

The angle computations via std::atan2 can be avoided by using only arithmetic operations inherent in the geometry of the problem. Listing 2 shows C++ code for a new less-than comparison function that is based solely on arithmetic operations and numeric comparisons. This allows the type T of listing 1 to be an exact rational type. The consequence is that the output indices of the sorting are theoretically correct. It is

possible to choose T to be a floating-point type, but then rounding errors in the floating-point arithmetic can lead to theoretically incorrect sorting.

Listing 2. C++ code for a less-than comparison for sorting vectors. The Sort function of listing 1 is valid for this implementation, but the less-than comparison function is different and uses only arithmetic operations. The code assumes $(0, 0)$ is not in the set of vectors. The if-then logic is a direct implementation of table 1.

```cpp
template <typename T>
bool LessThanByGeometry(SortObject<T> const& object0, SortObject<T> const& object1)
{
    T const& x0 = object0.W[0], y0 = object0.W[1];
    T const& x1 = object1.W[0], y1 = object1.W[1];
    T const zero = static_cast<T>(0);

    if (y0 < zero)
    {
        if (y1 < zero)
        {
            T c = x0 * y1 - x1 * y0;
            if (c > zero)
            {
                return true;
            }
            else if (c < zero)
            {
                return false;
            }
            T left = (x0 - x1) * (x0 + x1);
            T right = (y1 - y0) * (y1 + y0);
            return left < right;
        }
        else if (y1 > zero)
        {
            return true;
        }
        else  // y1 == zero
        {
            return true;
        }
    }
    else if (y0 > zero)
    {
        if (y1 < zero)
        {
            return false;
        }
        else if (y1 > zero)
        {
            T c = x0 * y1 - x1 * y0;
            if (c > zero)
            {
                return true;
            }
            else if (c < zero)
            {
                return false;
            }
            T left = (x0 - x1) * (x0 + x1);
            T right = (y1 - y0) * (y1 + y0);
            return left < right;
        }
        else  // y1 == zero
        {
            return x1 < zero;
        }
    }
```

```
    else  // y0 == zero
    {
        if (y1 < zero)
        {
            return false;
        }
        else if (y1 > zero)
        {
            return x0 > zero;
        }
        else  // y1 == zero
        {
            return (x1 < zero && x1 < x0) || (x0 > zero && x1 > x0);
        }
    }
}
```

The comparison $x_0^2+y_0^2 = s_0^2 < s_1^2 = x_1^2+y_1^2$ uses 4 multiplications, 2 additions and and a comparison of two numbers. Instead, the comparison is reformulated as $(x_0-x_1)(x_0+x_1) = x_0^2-x_1^2 < y_1^2-y_0^2 = (y_1-y_0)(y_1-y_0)$ which requires 2 multiplications, 4 additions and a comparison between two numbers. There is effectively no performance difference when T is a floating-point type. However, when T is an exact rational type implemented in software, multiplications are more expensive than additions. The reformulation is used for a performance improvement in this case.

Listing 2 can be modified so that the order of the test blocks defers arithmetic computations as late as possible in hopes of inexpensive early-out tests. Listing 3 shows the C++ code.

Listing 3. C++ code for a less-than comparison for sorting vectors with attempts at early returns to avoid arithemtic computations.

```
template <typename T>
bool LessThanByGeometry(SortObject<T> const& object0, SortObject<T> const& object1)
{
    T const& x0 = object0.W[0], y0 = object0.W[1];
    T const& x1 = object1.W[0], y1 = object1.W[1];
    T const zero = static_cast<T>(0);
    if (y0 < zero && y1 >= zero) { return true; }
    if (y1 < zero && y0 >= zero) { return false; }
    if (y0 > zero && y1 == zero) { return x1 < zero; }
    if (y1 > zero && y0 == zero) { return x0 > zero; }
    if (y0 == zero && y1 == zero) { return (x1 < zero && x1 < x0) || (x0 > zero && x1 > x0); }
    T c = x0 * y1 - x1 * y0;
    if (c > zero) { return true; }
    if (c < zero) { return false; }
    return (x0 - x1) * (x0 + x1) < (y1 - y0) * (y1 + y0);  // c == 0, s0 < s1
}
```

All comparisons in LessThanByGeometry are sign tests, so it is also possible to implement the algorithm using a blend of interval arithmetic and rational arithmetic. An expression is computed using interval arithmetic (floating-point computations with round-up and round-down). If the result has the theoretically correct sign, the execution can proceed using it. However, if the interval arithmetic produces a number for which the sign is not determined, the expression is recomputed using rational arithmetic in order to obtain the theoretically correct sign.

References

[1] Wikipedia. atan2.
https://en.wikipedia.org/wiki/Atan2.
accessed February 20, 2023.

[2] Wolfram Research, Inc. *Mathematica 13.2.1*. Wolfram Research, Inc., Champaign, Illinois, 2023.