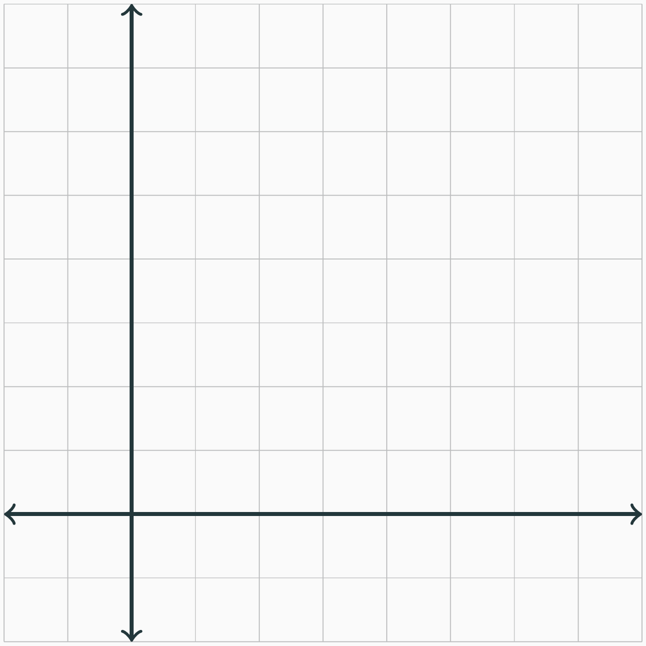
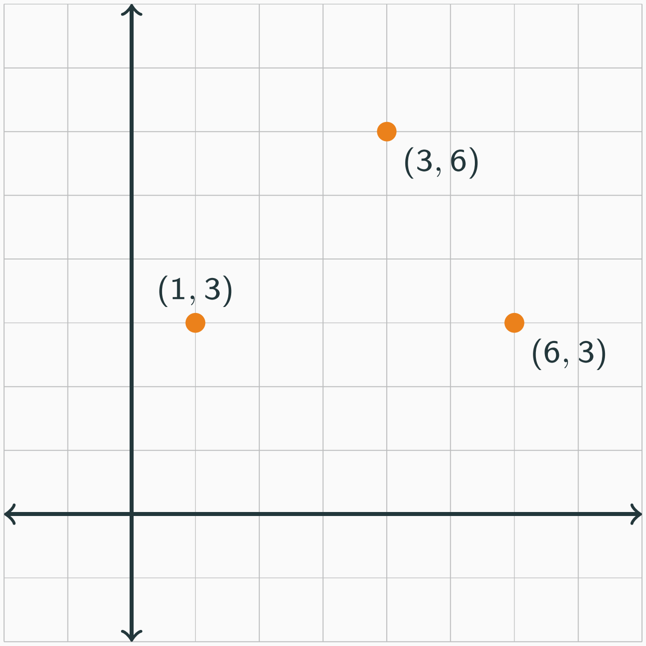
- Geometry
- Computational geometry

Geometry

Points and vectors

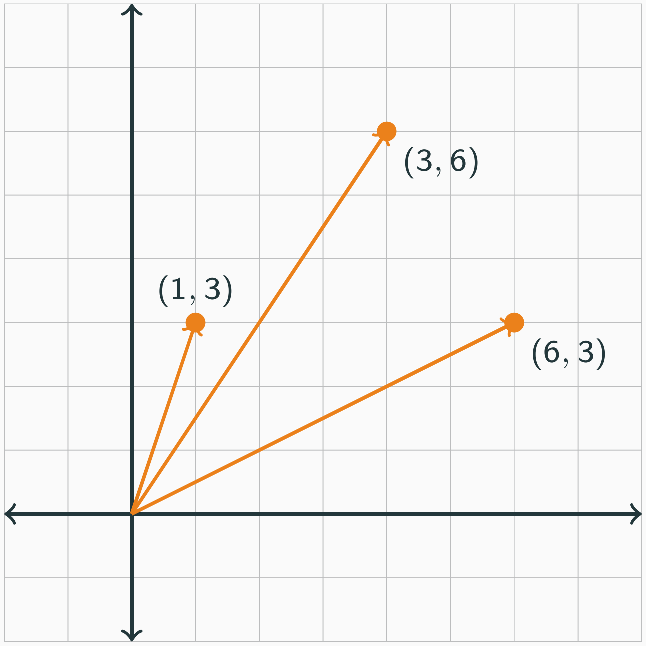


Points and vectors



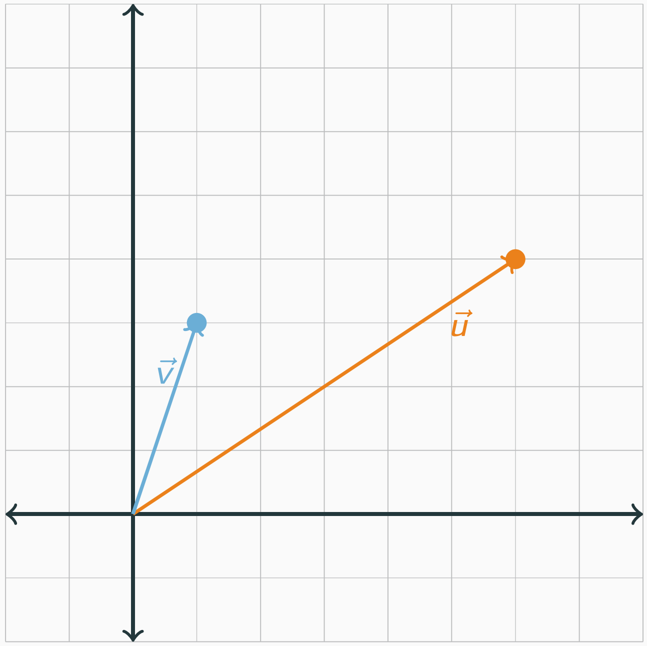
- Points are represented by a pair of numbers, (x, y).

Points and vectors

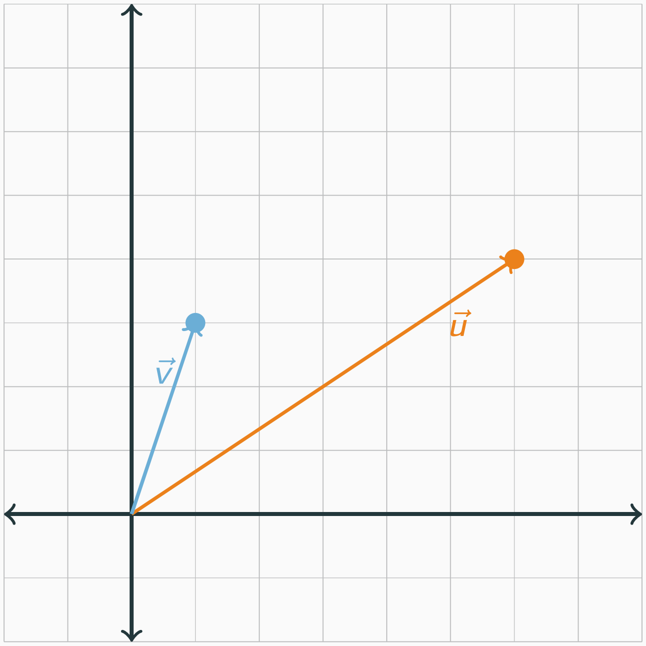


- Points are represented by a pair of numbers, (x, y).
- Vectors are represented in the same way.
- Thinking of points as vectors allows us to do many things.

Points and vectors



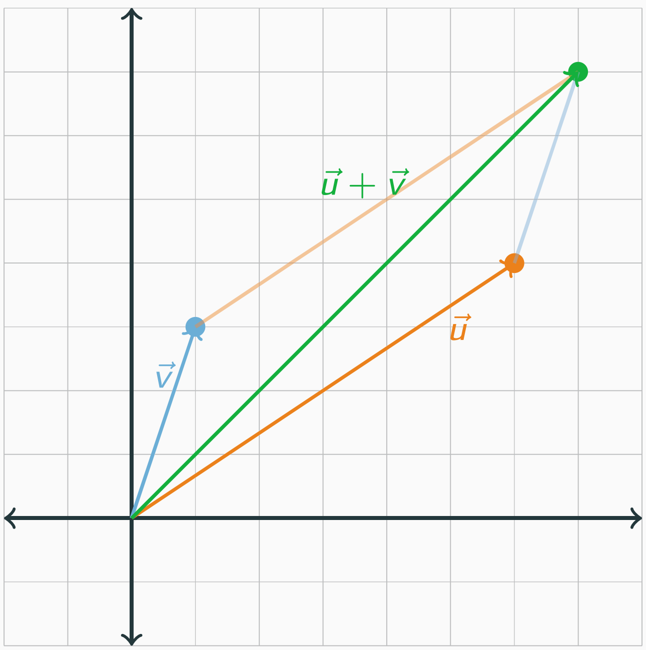
Points and vectors



- Simplest operation, addition is defined as

$$\begin{pmatrix} x_0 \\ y_0 \end{pmatrix} + \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} x_0 + x_1 \\ y_0 + y_1 \end{pmatrix}$$

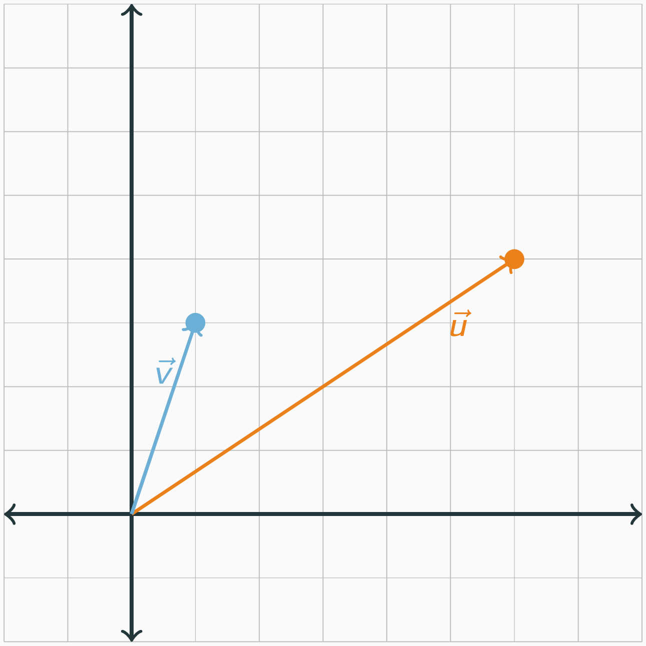
Points and vectors



- Simplest operation, addition is defined as

$$\begin{pmatrix} x_0 \\ y_0 \end{pmatrix} + \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} x_0 + x_1 \\ y_0 + y_1 \end{pmatrix}$$

Points and vectors



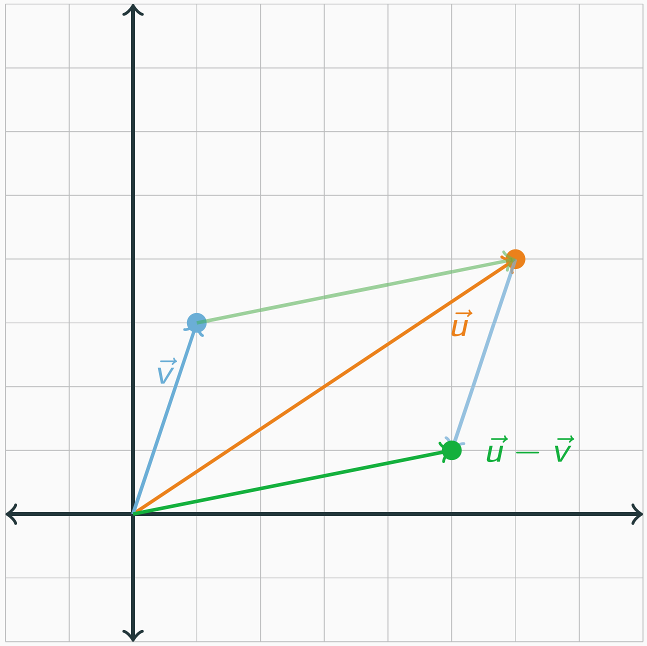
- Simplest operation, addition is defined as

$$\begin{pmatrix} x_0 \\ y_0 \end{pmatrix} + \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} x_0 + x_1 \\ y_0 + y_1 \end{pmatrix}$$

- Subtraction is defined in the same manner

$$\begin{pmatrix} x_0 \\ y_0 \end{pmatrix} - \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} x_0 - x_1 \\ y_0 - y_1 \end{pmatrix}$$

Points and vectors



- Simplest operation, addition is defined as

$$\begin{pmatrix} x_0 \\ y_0 \end{pmatrix} + \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} x_0 + x_1 \\ y_0 + y_1 \end{pmatrix}$$

- Subtraction is defined in the same manner

$$\begin{pmatrix} x_0 \\ y_0 \end{pmatrix} - \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} x_0 - x_1 \\ y_0 - y_1 \end{pmatrix}$$

Points and vectors

```
struct point {
    double x, y;
    point(double _x, double _y) {
        x = _x, y = _y;
    }

    point operator+(const point &oth){
        return point(x + oth.x, y + oth.y);
    }

    point operator-(const point &oth){
        return point(x - oth.x, y - oth.y);
    }
};
```

Points and vectors

. . . or we could use the `complex<double>` class.

```
typedef complex<double> point;
```

Points and vectors

. . . or we could use the `complex<double>` class.

```
typedef complex<double> point;
```

The `complex` class in `C++` and `Java` has methods defined for

- Addition
- Subtraction
- Multiplication by a scalar
- Length
- Trigonometric functions

Points and vectors

Complex numbers have the real part and the imaginary part. Can be thought of as vectors or points on the complex plane.

Points and vectors

Complex numbers have the real part and the imaginary part. Can be thought of as vectors or points on the complex plane.

- `double real(p)` returns the real part, in our case, the x value of `p`

Points and vectors

Complex numbers have the real part and the imaginary part. Can be thought of as vectors or points on the complex plane.

- `double real(p)` returns the real part, in our case, the x value of `p`
- `double imag(p)` returns the imaginary part, y value of `p`.

Points and vectors

Complex numbers have the real part and the imaginary part. Can be thought of as vectors or points on the complex plane.

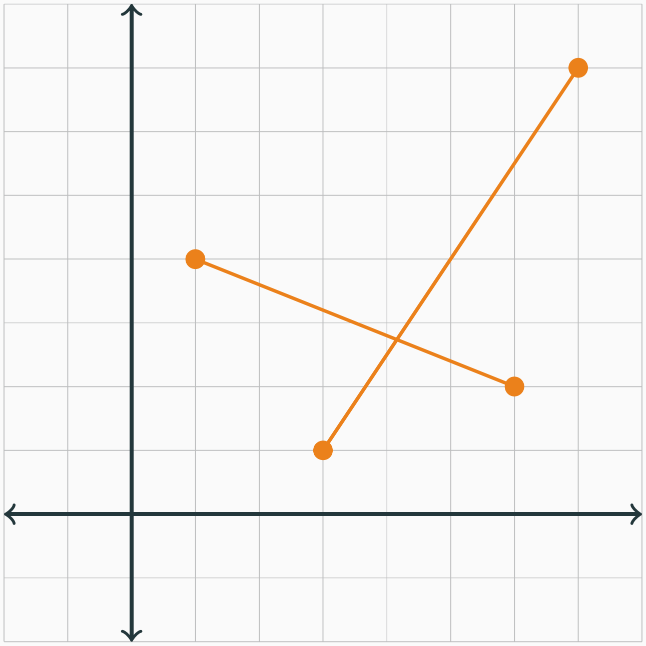
- `double real(p)` returns the real part, in our case, the x value of `p`
- `double imag(p)` returns the imaginary part, y value of `p`.
- `double abs(p)` returns the absolute value of the complex number, the length of the vector.

Points and vectors

Complex numbers have the real part and the imaginary part. Can be thought of as vectors or points on the complex plane.

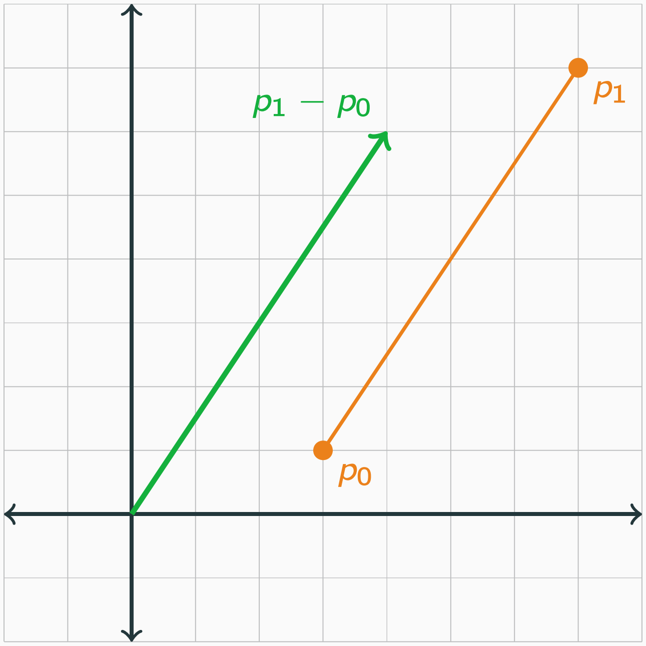
- `double real(p)` returns the real part, in our case, the x value of `p`
- `double imag(p)` returns the imaginary part, y value of `p`.
- `double abs(p)` returns the absolute value of the complex number, the length of the vector.
- `double sin(p)`, `double cos(p)`, `double tan(p)`, trigonometric functions.

Lines and line segments



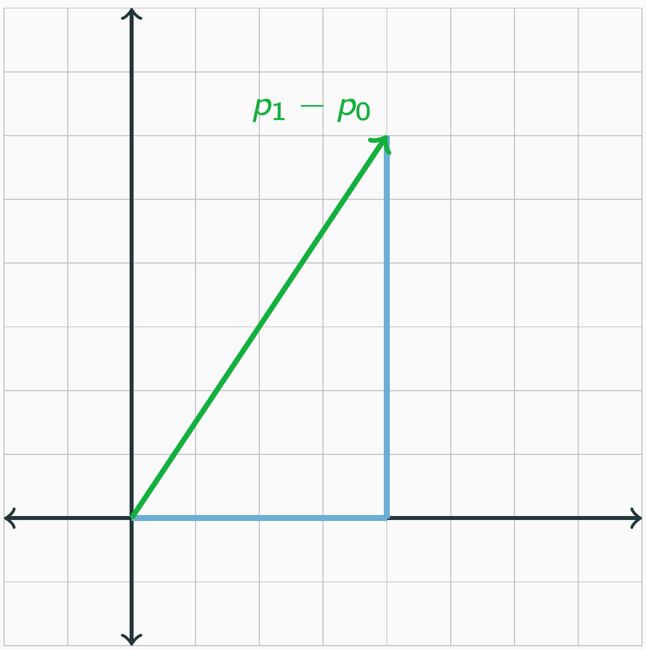
- Line segments are represented by a pair of points, $((x_0, y_0), (x_1, y_1))$.

Lines and line segments



- Line segments are represented by a pair of points, $((x_0, y_0), (x_1, y_1))$.
- Distance between two points is the length of the line segment or vector between the points.

Lines and line segments



- Line segments are represented by a pair of points, $((x_0, y_0), (x_1, y_1))$.
- Distance between two points is the length of the line segment or vector between the points.

$$
\begin{aligned}
d((x_0,&y_0),(x_1,y_1)) \\
&= |(x_1 - x_0, y_1 - y_0)| \\
&= \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2}
\end{aligned}
$$

Lines and line segments

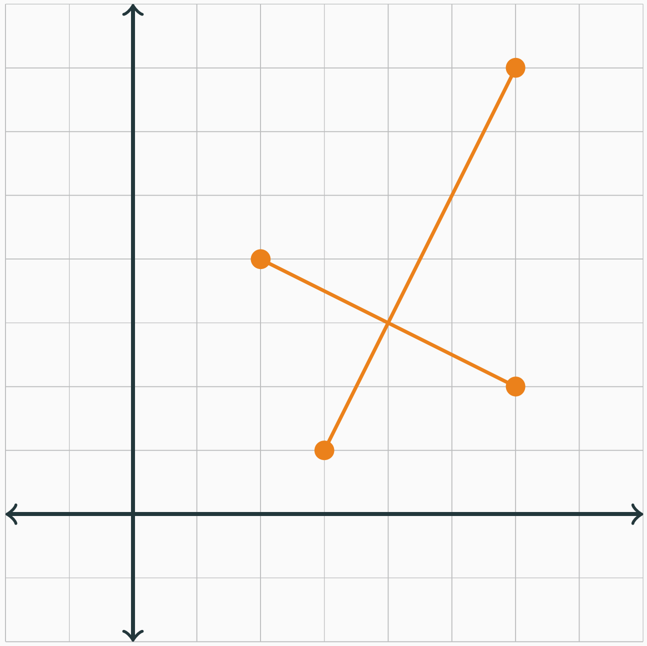
```
struct point {
    ...
    double distance(point oth = point(0,0)) const {
        return sqrt(pow(x - oth.x, 2.0)
                  + pow(y - oth.y, 2.0));
    }
    ...
}
```

Lines and line segments

```
struct point {
    ...
    double distance(point oth = point(0,0)) const {
        return sqrt(pow(x - oth.x, 2.0)
                  + pow(y - oth.y, 2.0));
    }
    ...
}
```

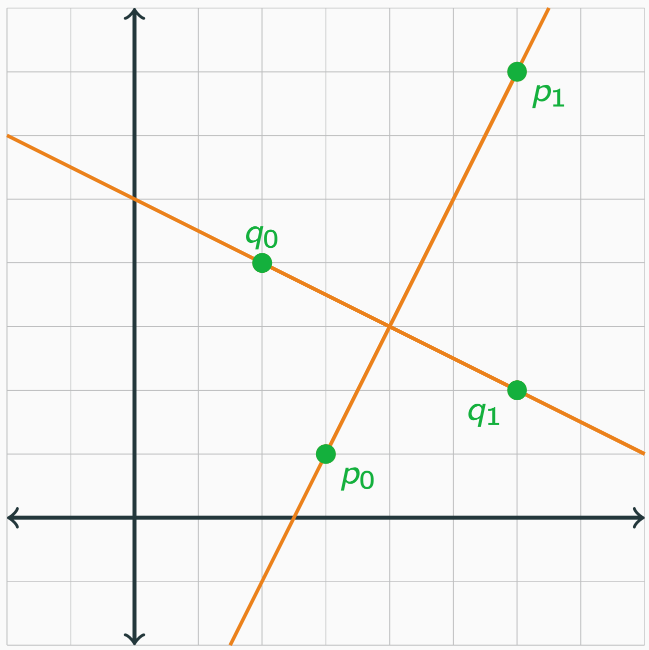
Or use the `abs` function with `complex<double>`.

Lines and line segments



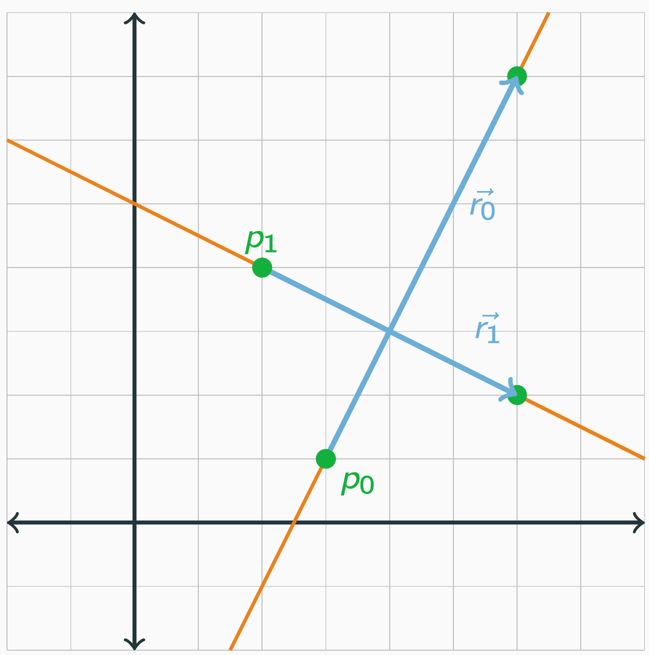
- Line representation same as line segments.

Lines and line segments



- Line representation same as line segments.
- Treat them as lines passing through the two points.

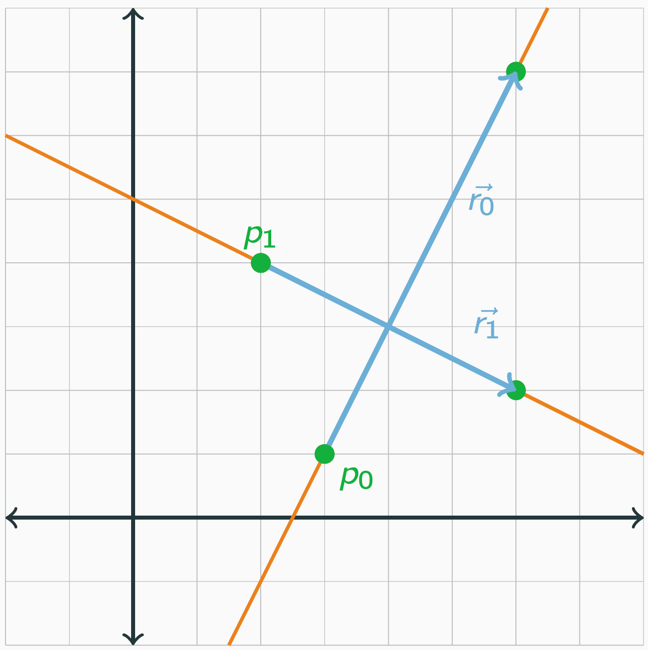
Lines and line segments



- Line representation same as line segments.
- Treat them as lines passing through the two points.
- Or as a point and a direction vector.

$$p + t \cdot \vec{r}$$

Lines and line segments



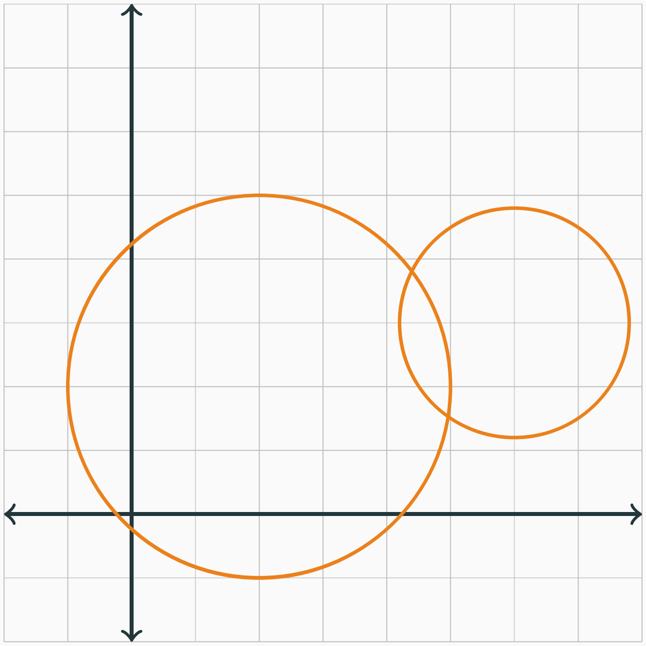
- Line representation same as line segments.
- Treat them as lines passing through the two points.
- Or as a point and a direction vector.

$$p + t \cdot \vec{r}$$

- Either way

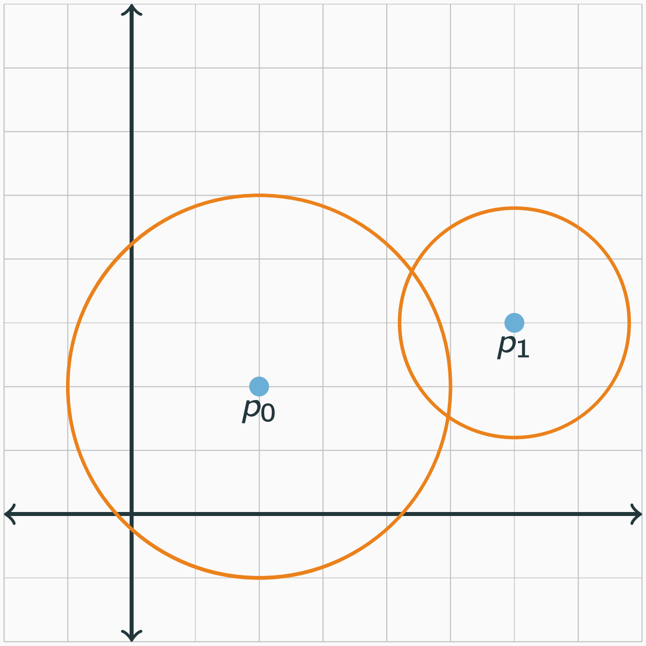
`pair<point,point>`

Circles



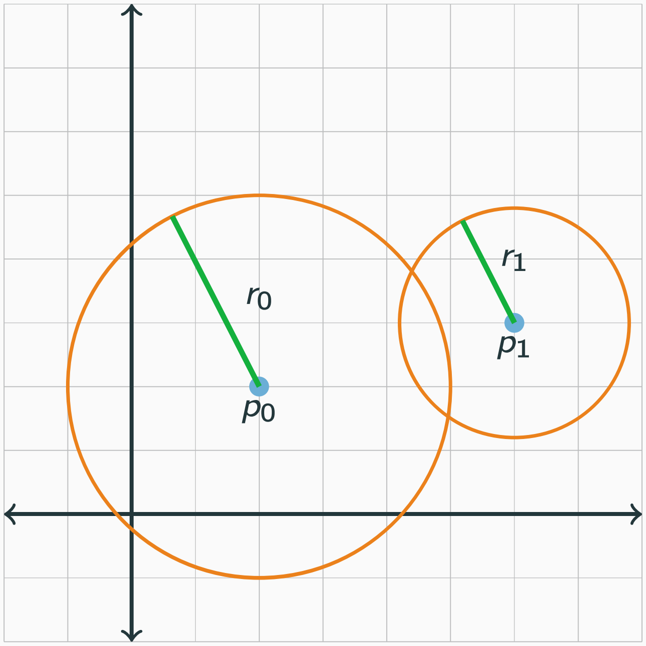
- Circles are very easy to represent.

Circles



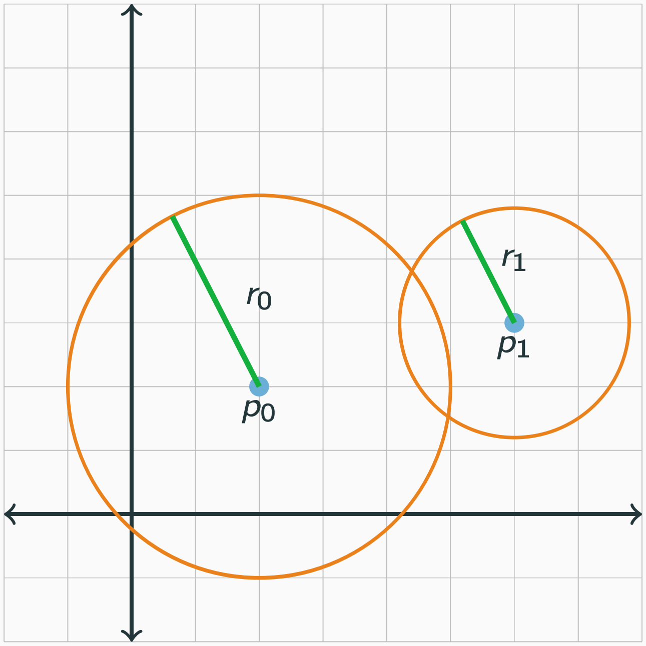
- Circles are very easy to represent.
- Center point $p = (x, y)$.

Circles



- Circles are very easy to represent.
- Center point $p = (x, y)$.
- And the radius r.

Circles



- Circles are very easy to represent.
- Center point $p = (x, y)$.
- And the radius r.

`pair<point,double>`

Dot product

Given two vectors

$$\vec{u} = \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} \qquad \vec{v} = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}$$

the dot product of $\vec{u}$ and $\vec{v}$ is defined as

$$\begin{pmatrix} x_0 \\ y_0 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = x_0 \cdot x_1 + y_0 \cdot y_1$$

Dot product

Given two vectors

$$\vec{u} = \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} \qquad \vec{v} = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}$$

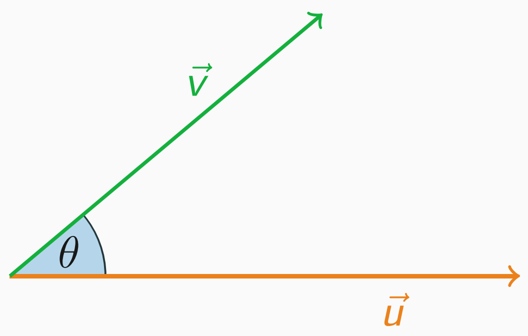
the dot product of $\vec{u}$ and $\vec{v}$ is defined as

$$\begin{pmatrix} x_0 \\ y_0 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = x_0 \cdot x_1 + y_0 \cdot y_1$$

Which in geometric terms is

$$\vec{u} \cdot \vec{v} = |\vec{u}||\vec{v}| \cos \theta$$

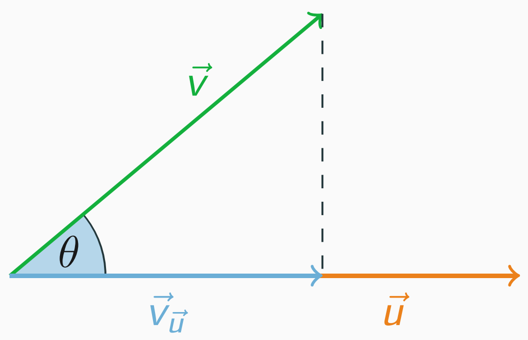
Dot product



- Allows us to calculate the angle between $\vec{u}$ and $\vec{v}$.

$$\theta = \arccos\left(\frac{\vec{u}\cdot\vec{v}}{|\vec{u}||\vec{v}|}\right)$$

Dot product



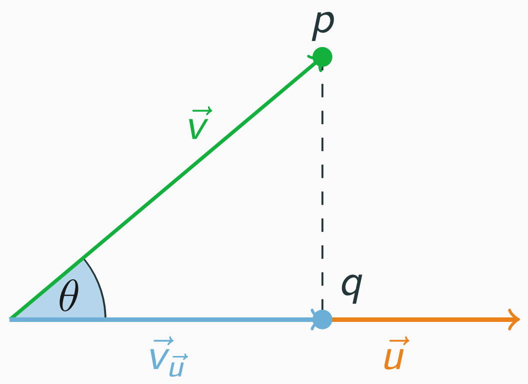
- Allows us to calculate the angle between $\vec{u}$ and $\vec{v}$.

$$\theta = \arccos\left(\frac{\vec{u}\cdot\vec{v}}{|\vec{u}||\vec{v}|}\right)$$

- And the projection of $\vec{v}$ onto $\vec{u}$.

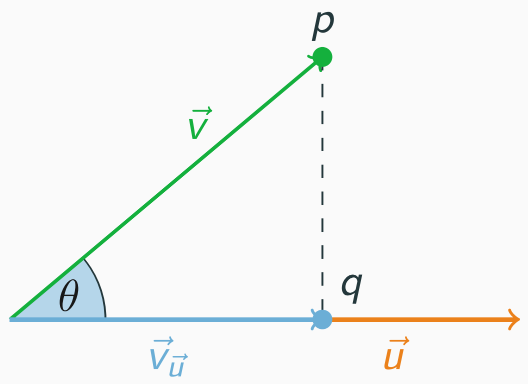
$$\vec{v}_{\vec{u}} = \left(\frac{\vec{u}\cdot\vec{v}}{|u|^2}\right)\vec{u}$$

Dot product



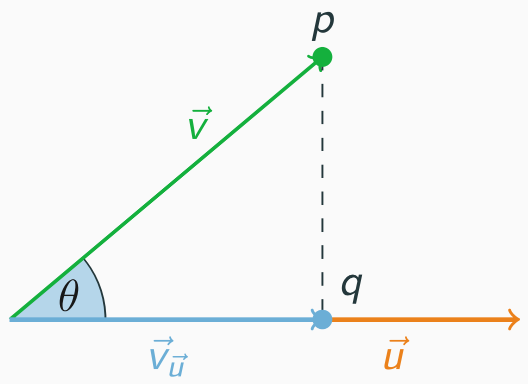
- The closest point on $\vec{u}$ to p is q.

Dot product



- The closest point on $\vec{u}$ to p is q.
- The distance from p to $\vec{u}$ is the distance from p to q.

Dot product



- The closest point on $\vec{u}$ to p is q.
- The distance from p to $\vec{u}$ is the distance from p to q.
- Unless q is outside $\vec{u}$, then the closest point is either of the endpoints.

Dot product

Rest of the code will use the `complex` class.

```
#define P(p) const point &p
#define L(p0, p1) P(p0), P(p1)
double dot(P(a), P(b)) {
    return real(a) * real(b) + imag(a) * imag(b);
}
double angle(P(a), P(b), P(c)) {
    return acos(dot(b - a, c - b) / abs(b - a) / abs(c - b));
}
point closest_point(L(a, b), P(c), bool segment = false) {
    if (segment) {
        if (dot(b - a, c - b) > 0) return b;
        if (dot(a - b, c - a) > 0) return a;
    }
    double t = dot(c - a, b - a) / norm(b - a);
    return a + t * (b - a);
}
```

Cross product

Given two vectors

$$\vec{u} = \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} \qquad \vec{v} = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}$$

the cross product of $\vec{u}$ and $\vec{v}$ is defined as

$$\left| \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} \times \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} \right| = x_0 \cdot y_1 - y_0 \cdot x_1$$

Cross product

Given two vectors

$$\vec{u} = \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} \qquad \vec{v} = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}$$

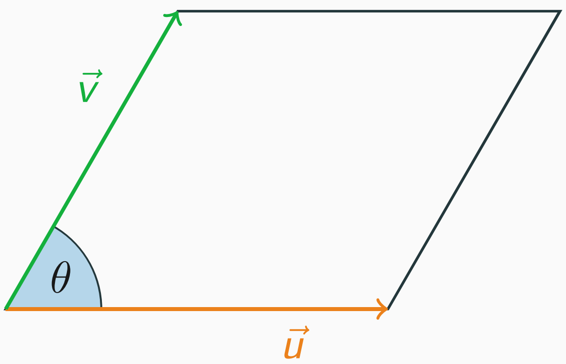
the cross product of $\vec{u}$ and $\vec{v}$ is defined as

$$\left| \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} \times \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} \right| = x_0 \cdot y_1 - y_0 \cdot x_1$$

Which in geometric terms is

$$|\vec{u} \times \vec{v}| = |\vec{u}||\vec{v}| \sin \theta$$

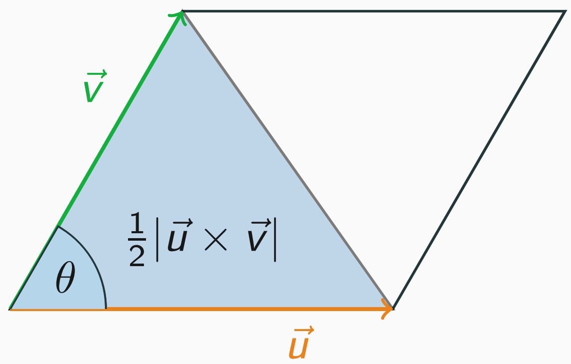
Cross product



- Allows us to calculate the area of the triangle formed by $\vec{u}$ and $\vec{v}$.

$$\frac{|\vec{u} \times \vec{v}|}{2}$$

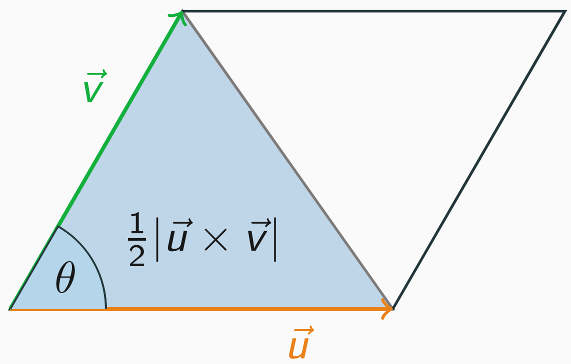
Cross product



- Allows us to calculate the area of the triangle formed by $\vec{u}$ and $\vec{v}$.

$$\frac{|\vec{u} \times \vec{v}|}{2}$$

Cross product



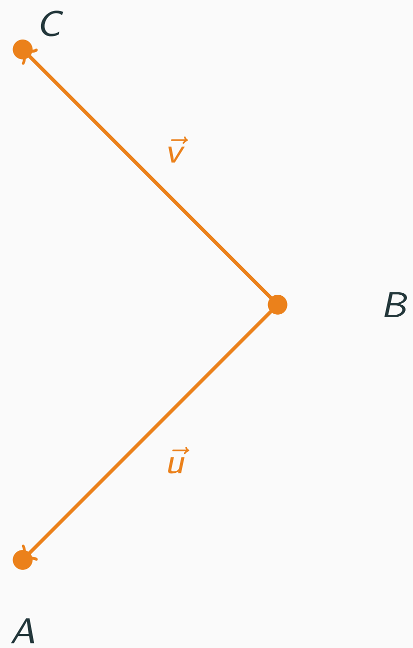
- Allows us to calculate the area of the triangle formed by $\vec{u}$ and $\vec{v}$.

$$\frac{|\vec{u} \times \vec{v}|}{2}$$

- And can tell us if the angle between $\vec{u}$ and $\vec{v}$ is positive or negative.

$$\begin{aligned}
|\vec{u} \times \vec{v}| < 0 \quad &\text{iff} \quad \theta < \pi \\
|\vec{u} \times \vec{v}| = 0 \quad &\text{iff} \quad \theta = \pi \\
|\vec{u} \times \vec{v}| > 0 \quad &\text{iff} \quad \theta > \pi
\end{aligned}$$

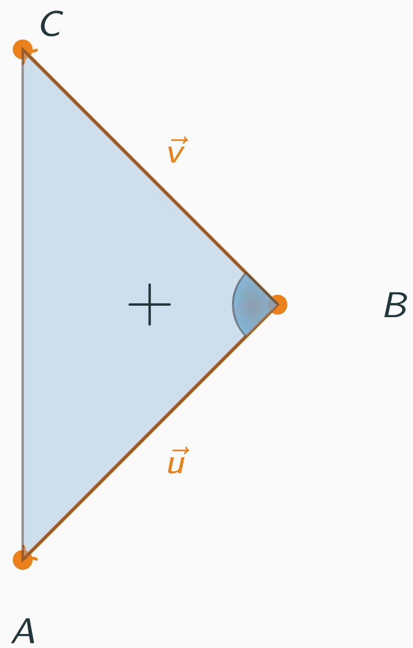
Counterclockwise



- Given three points A, B and C, we want to know if they form a counter-clockwise angle in that order.

$$A \rightarrow B \rightarrow C$$

Counterclockwise



- Given three points A, B and C, we want to know if they form a counter-clockwise angle in that order.

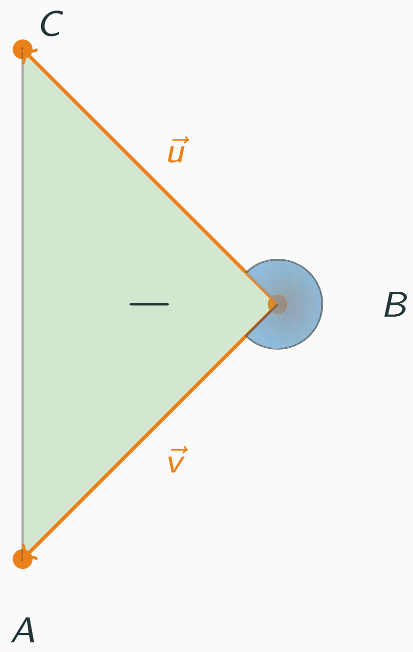
$$A \to B \to C$$

- We can examine the cross product of and the area of the triangle formed by

$$\vec{u} = B - C \quad \vec{v} = B - A$$

$$\vec{u} \times \vec{v} > 0$$

Counterclockwise



- The points in the reverse order do not form a counter clockwise angle.

$$C \to B \to A$$

- In the reverse order the vectors swap places

$$\vec{u} = B - A \quad \vec{v} = B - C$$
$$\vec{u} \times \vec{v} < 0$$

Counterclockwise

- The points in the reverse order do not form a counter clockwise angle.

$$C \to B \to A$$

- In the reverse order the vectors swap places

$$\vec{u} = B - A \quad \vec{v} = B - C$$
$$\vec{u} \times \vec{v} < 0$$

- If the points A, B and C are on the same line, then the area will be 0.

Counterclockwise

```
double cross(P(a), P(b)) {
    return real(a)*imag(b) - imag(a)*real(x);
}
double ccw(P(a), P(b), P(c)) {
    return cross(b - a, c - b);
}
bool collinear(P(a), P(b), P(c)) {
    return abs(ccw(a, b, c)) < EPS;
}
```

Intersections

Very common task is to find the intersection of two lines or line segments.

Intersections

Very common task is to find the intersection of two lines or line segments.

- Given a pair of points (x_0, y_0), (x_1, y_1), representing a line we want to start by obtaining the form $Ax + By = C$.

Intersections

Very common task is to find the intersection of two lines or line segments.

- Given a pair of points (x_0, y_0), (x_1, y_1), representing a line we want to start by obtaining the form $Ax + By = C$.
- We can do so by setting

$$A = y_1 - y_0$$
$$B = x_0 - x_1$$
$$C = A \cdot x_0 + B \cdot y_1$$

Intersections

Very common task is to find the intersection of two lines or line segments.

- Given a pair of points (x_0, y_0), (x_1, y_1), representing a line we want to start by obtaining the form $Ax + By = C$.
- We can do so by setting

$$
\begin{aligned}
A &= y_1 - y_0 \\
B &= x_0 - x_1 \\
C &= A \cdot x_0 + B \cdot y_1
\end{aligned}
$$

- If we have two lines given by such equations, we simply need to solve for the two unknowns, x and y.

Intersections

For two lines

$$A_0x + B_0y = C_0$$
$$A_1x + B_1y = C_1$$

The intersection point is

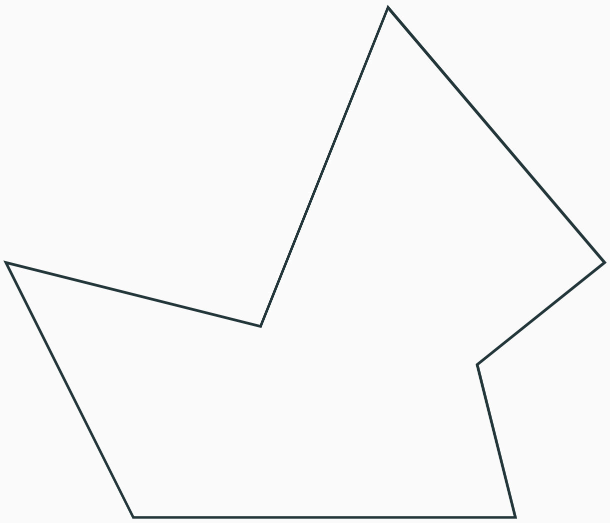
$$x = \frac{(B_1 \cdot C_0 - B_0 \cdot C_1)}{D}$$
$$y = \frac{(A_0 \cdot C_1 - A_1 \cdot C_0)}{D}$$

Where

$$D = A_0 \cdot B_1 - A_1 \cdot B_0$$

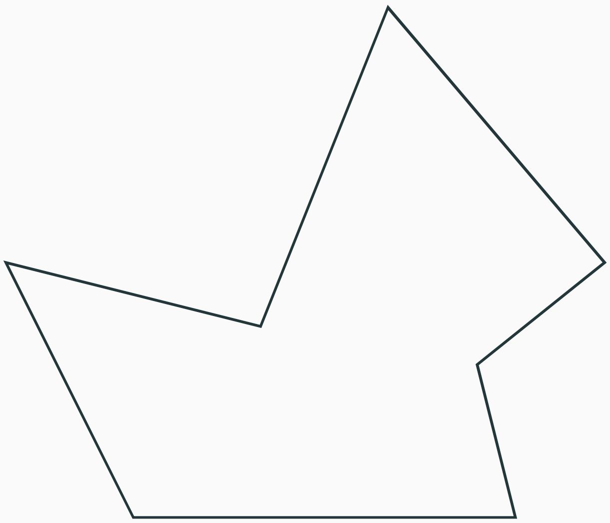
Computational Geometry

Polygons



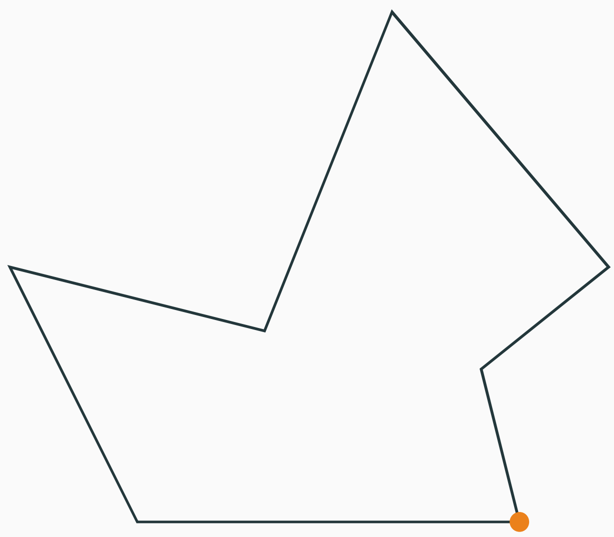
- Polygons are represented by a list of points in the order representing the edges.

Polygons



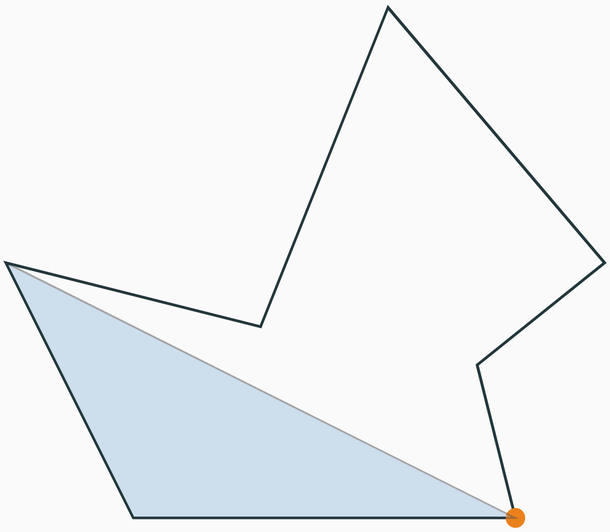
- Polygons are represented by a list of points in the order representing the edges.
- To calculate the area

Polygons



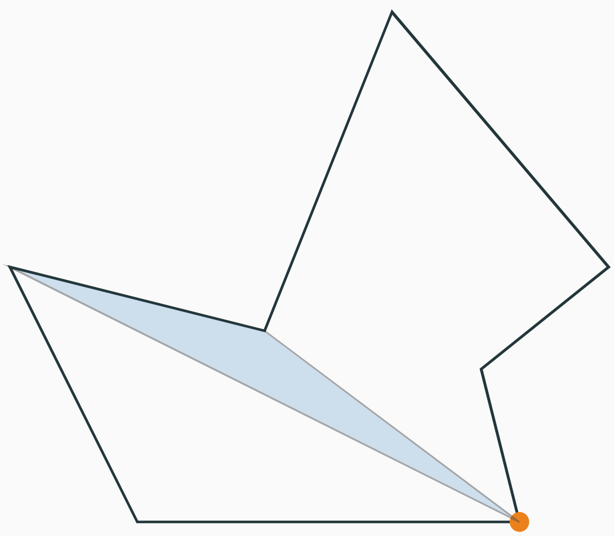
- Polygons are represented by a list of points in the order representing the edges.
- To calculate the area
  - We pick one starting point.

Polygons



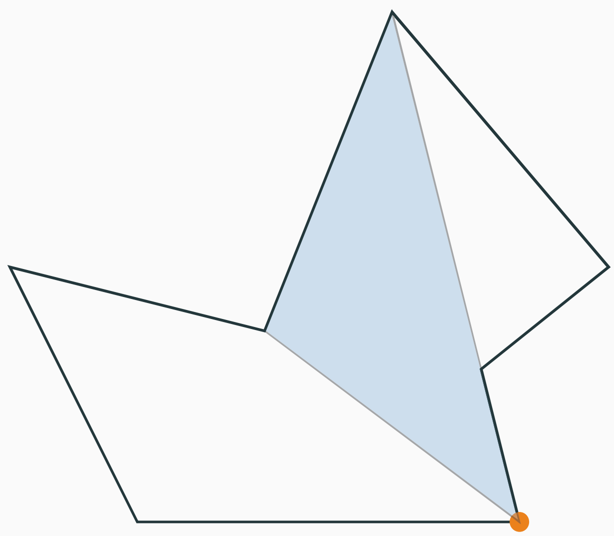
- Polygons are represented by a list of points in the order representing the edges.
- To calculate the area
  - We pick one starting point.
  - Go through all the other adjacent pair of points and sum the area of the triangulation.

Polygons



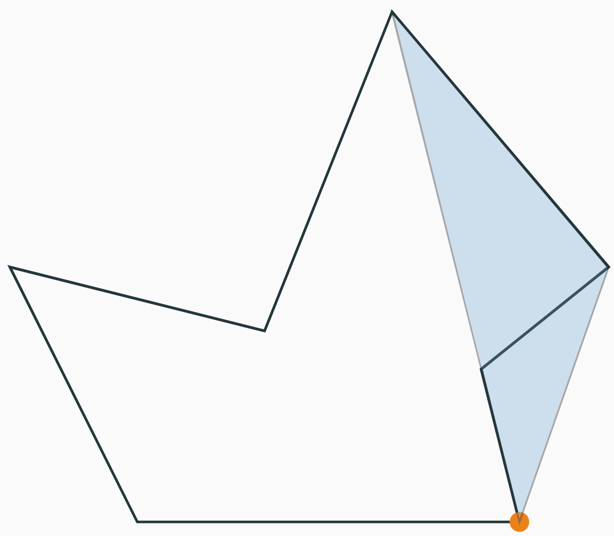
- Polygons are represented by a list of points in the order representing the edges.
- To calculate the area
  - We pick one starting point.
  - Go through all the other adjacent pair of points and sum the area of the triangulation.

Polygons



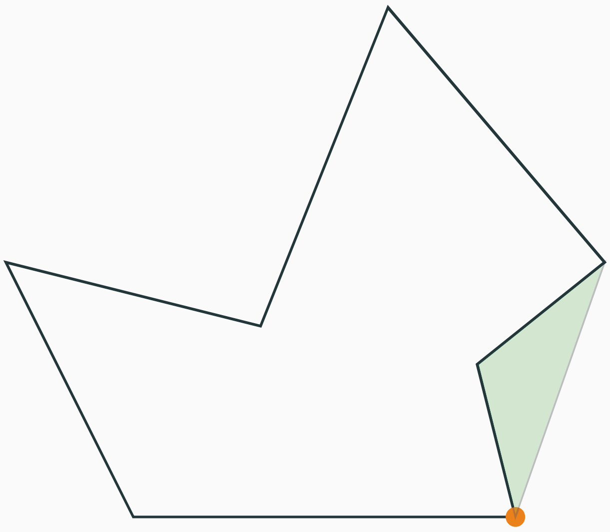
- Polygons are represented by a list of points in the order representing the edges.
- To calculate the area
  - We pick one starting point.
  - Go through all the other adjacent pair of points and sum the area of the triangulation.

Polygons



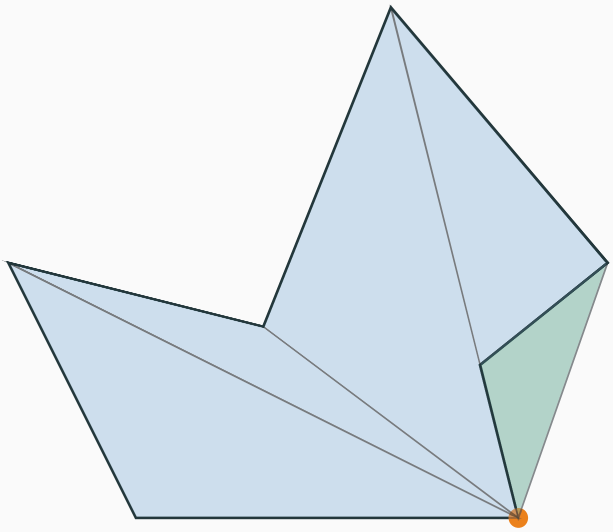
- Polygons are represented by a list of points in the order representing the edges.
- To calculate the area
  - We pick one starting point.
  - Go through all the other adjacent pair of points and sum the area of the triangulation.

Polygons



- Polygons are represented by a list of points in the order representing the edges.
- To calculate the area
  - We pick one starting point.
  - Go through all the other adjacent pair of points and sum the area of the triangulation.
  - Even if we sum up area outside the polygon, due to the cross product, it is subtracted later.

Polygons



- Polygons are represented by a list of points in the order representing the edges.
- To calculate the area
  - We pick one starting point.
  - Go through all the other adjacent pair of points and sum the area of the triangulation.
  - Even if we sum up area outside the polygon, due to the cross product, it is subtracted later.

Convex hull

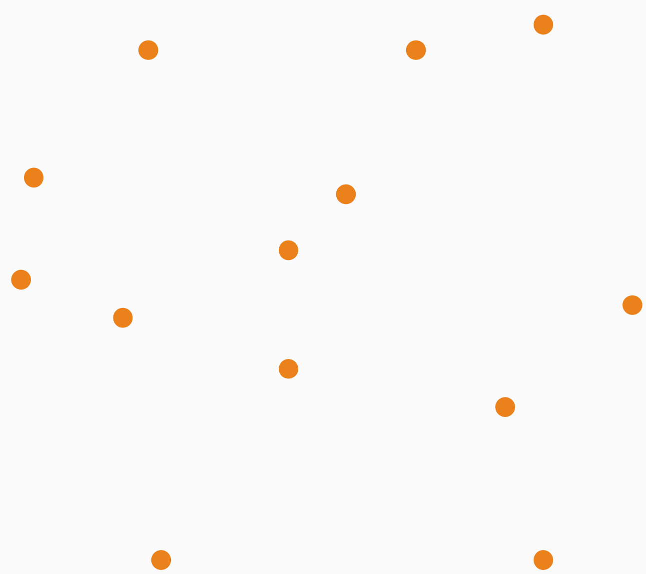
- Given a set of points, we want to find the convex hull of the points.

Convex hull

- Given a set of points, we want to find the convex hull of the points.
- The convex hull of points can be visualized as the shape formed by a rubber band around the set of points.

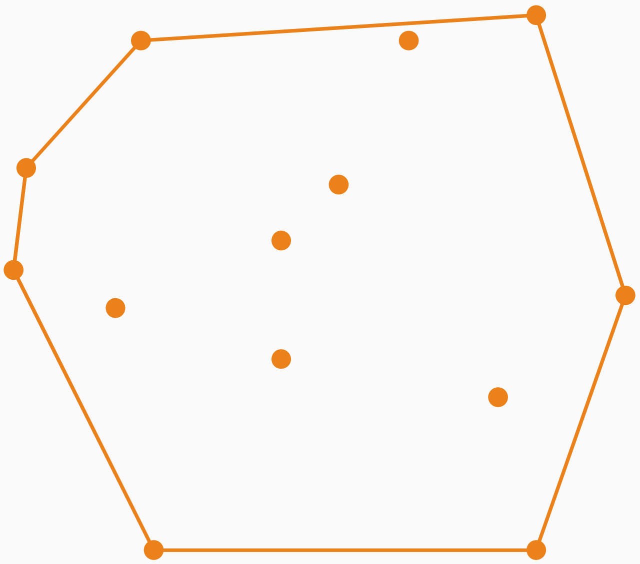
Convex hull

- Given a set of points, we want to find the convex hull of the points.
- The convex hull of points can be visualized as the shape formed by a rubber band around the set of points.



Convex hull

- Given a set of points, we want to find the convex hull of the points.
- The convex hull of points can be visualized as the shape formed by a rubber band around the set of points.



Convex hull

Graham scan:

Convex hull

Graham scan:

- Pick the point p_0 with the lowest y coordinate.

Convex hull

Graham scan:

- Pick the point p_0 with the lowest y coordinate.
- Sort all the points by polar angle with p_0.

Convex hull

Graham scan:

- Pick the point p_0 with the lowest y coordinate.
- Sort all the points by polar angle with p_0.
- Iterate through all the points

Convex hull

Graham scan:

- Pick the point p_0 with the lowest y coordinate.
- Sort all the points by polar angle with p_0.
- Iterate through all the points
- If the current point forms a clockwise angle with the last two points, remove last point from the convex set.

Convex hull

Graham scan:

- Pick the point p_0 with the lowest y coordinate.
- Sort all the points by polar angle with p_0.
- Iterate through all the points
- If the current point forms a clockwise angle with the last two points, remove last point from the convex set.
- Otherwise, add the current point to the convex set.

Convex hull

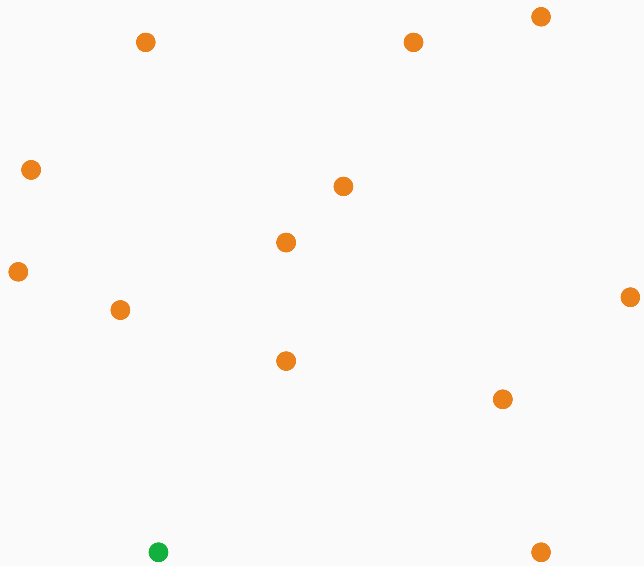
Graham scan:

- Pick the point p_0 with the lowest y coordinate.
- Sort all the points by polar angle with p_0.
- Iterate through all the points
- If the current point forms a clockwise angle with the last two points, remove last point from the convex set.
- Otherwise, add the current point to the convex set.

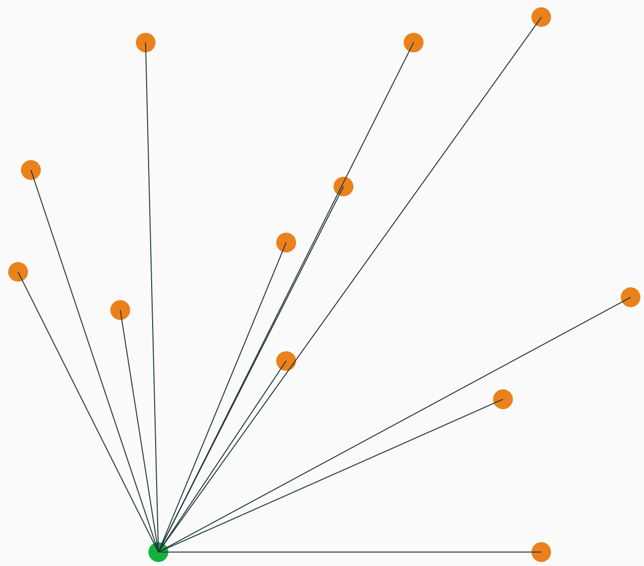
Time complexity $O(N \log N)$.

Convex hull

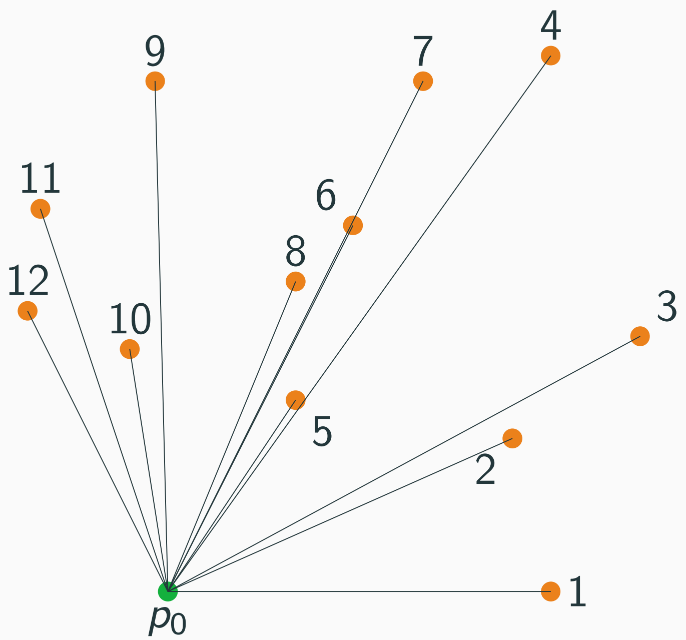
Convex hull



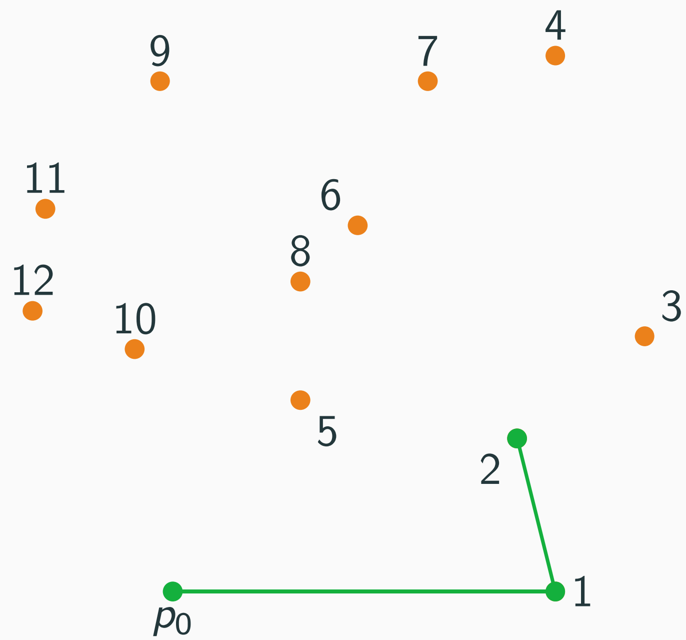
Convex hull



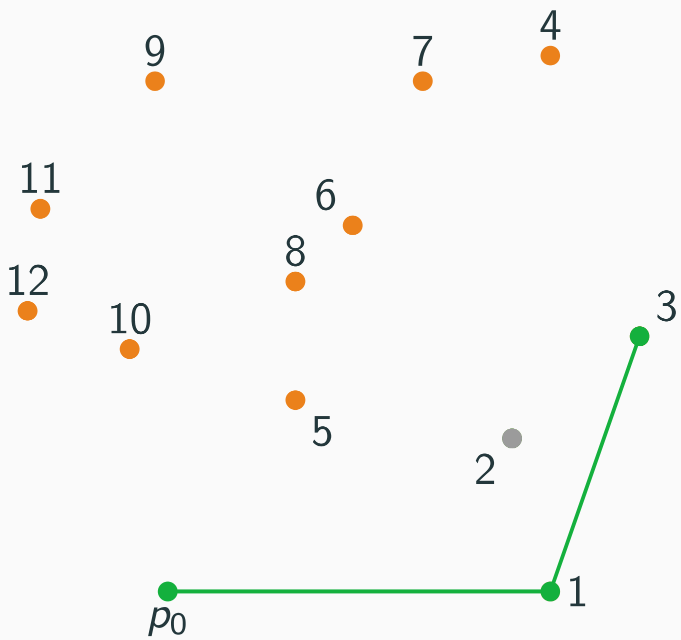
Convex hull



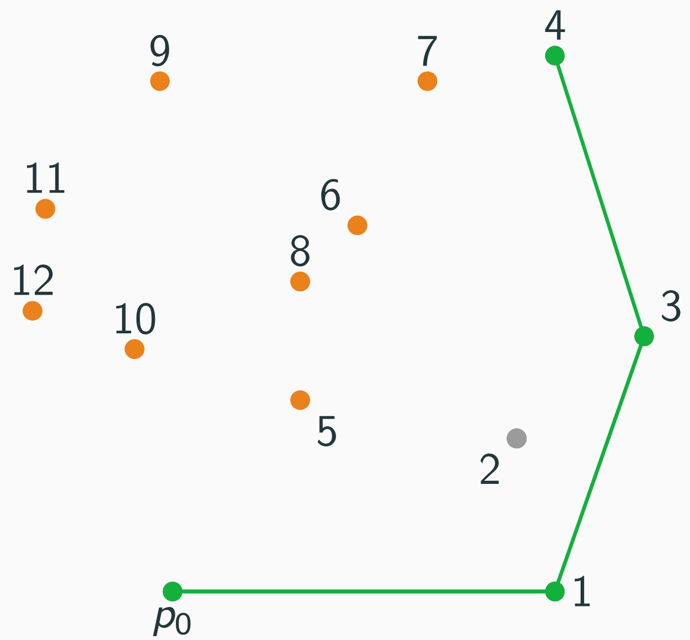
Convex hull



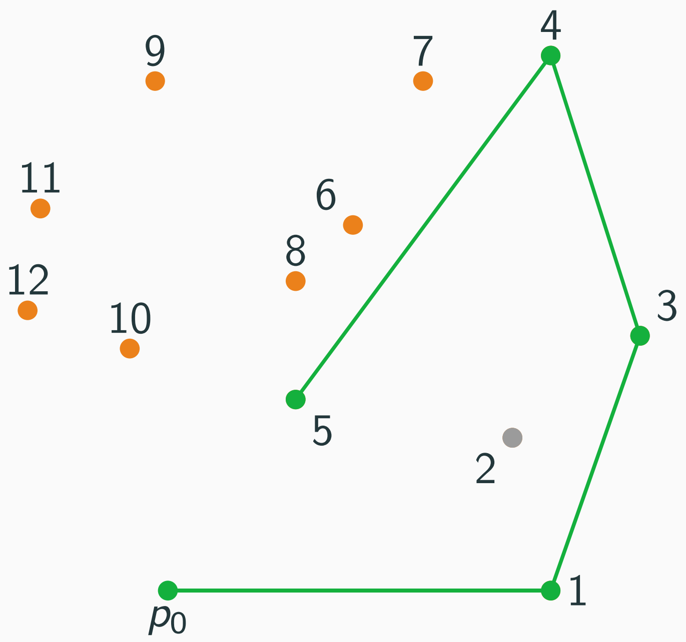
Convex hull



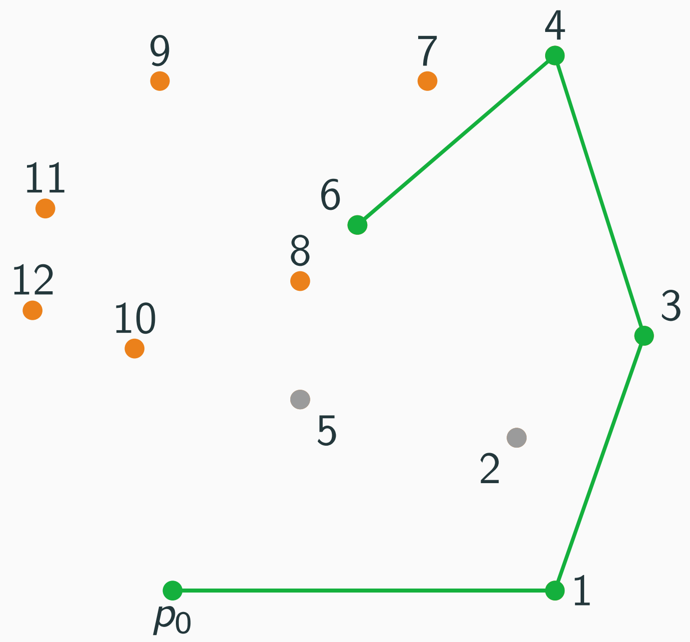
Convex hull



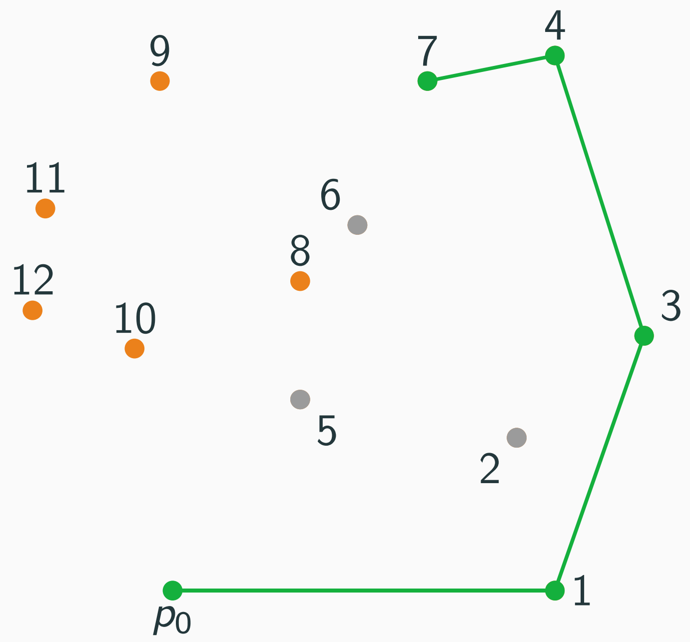
Convex hull



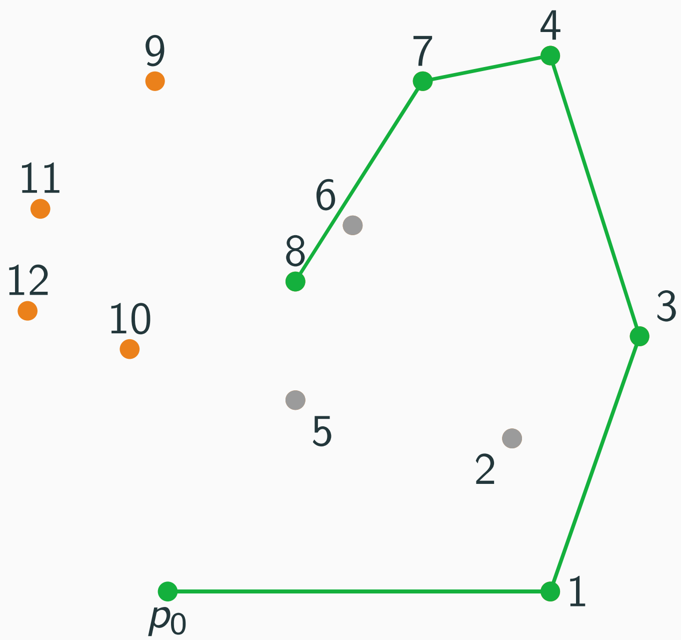
Convex hull



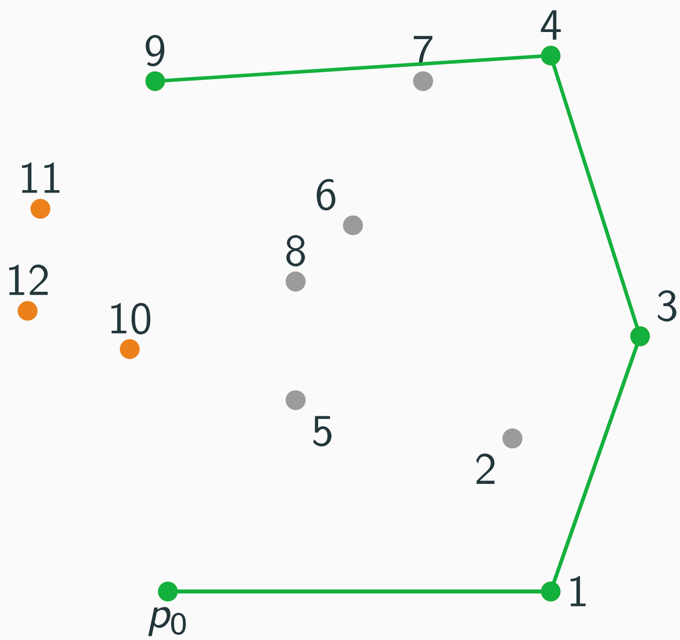
Convex hull



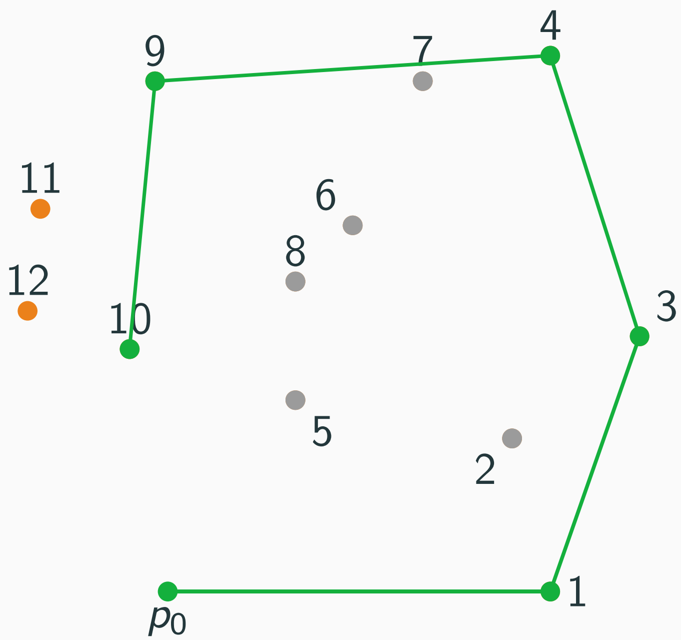
Convex hull



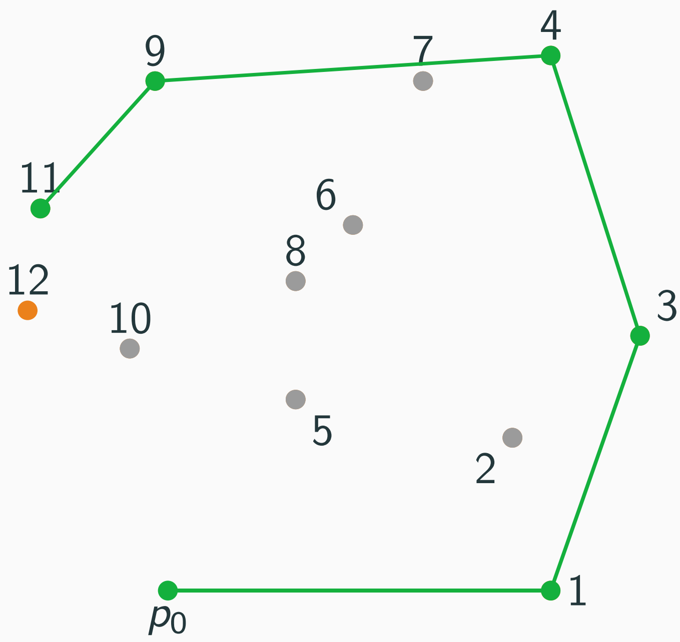
Convex hull



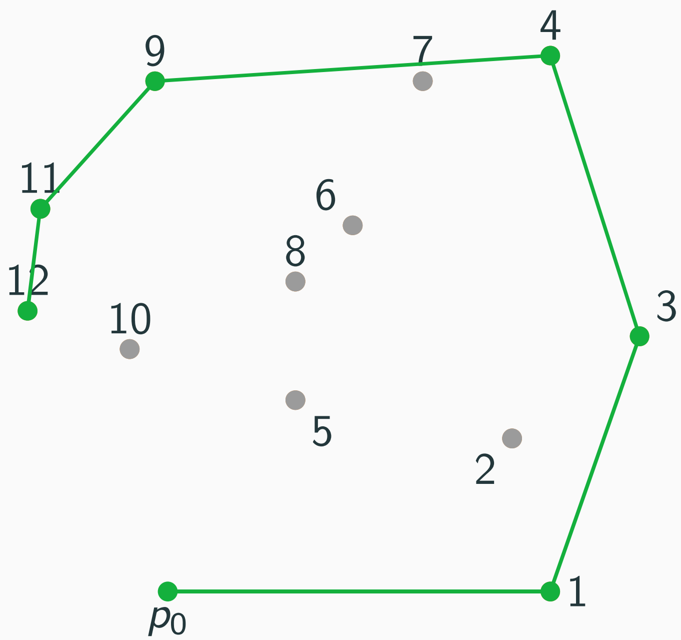
Convex hull



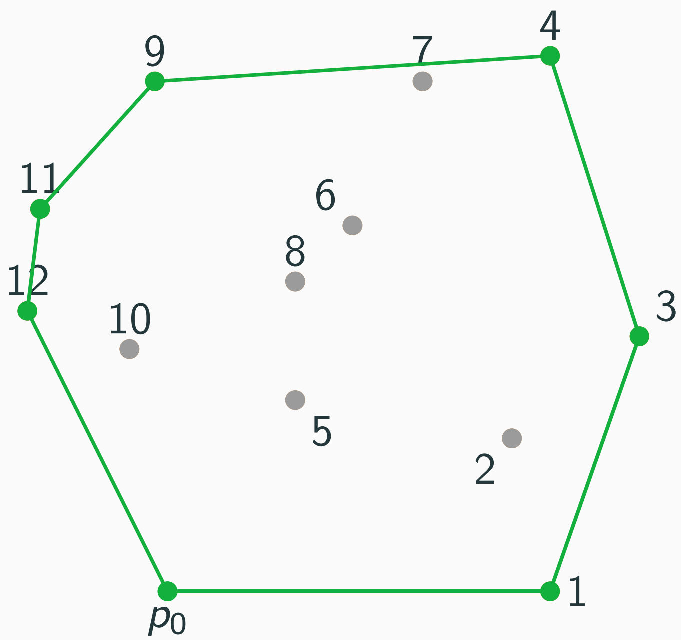
Convex hull



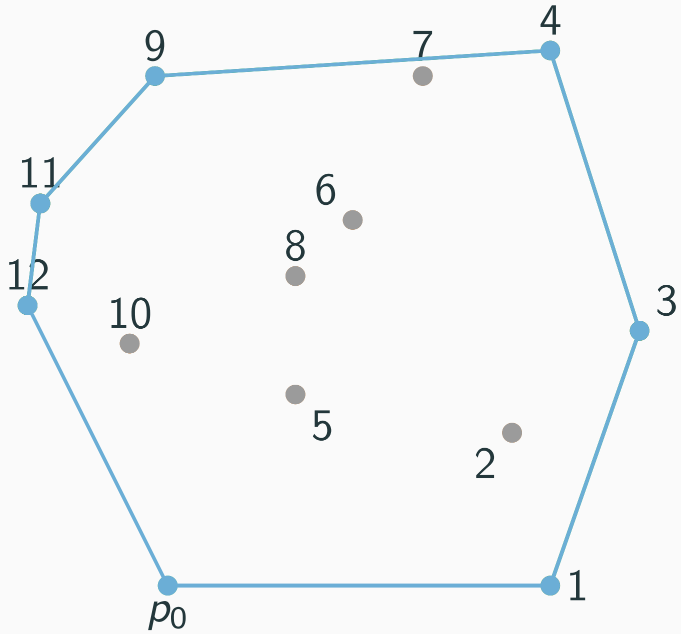
Convex hull



Convex hull



Convex hull



Convex hull

```
point hull[MAXN];
int convex_hull(vector<point> p) {
    int n = size(p), l = 0;
    sort(p.begin(), p.end(), cmp);
    for (int i = 0; i < n; i++) {
        if (i > 0 && p[i] == p[i - 1])
            continue;
        while (l >= 2 && ccw(hull[l - 2], hull[l - 1], p[i]) >= 0)
            l--;
        hull[l++] = p[i];
    }
    int r = l;
    for (int i = n - 2; i >= 0; i--) {
        if (p[i] == p[i + 1])
            continue;
        while (r - l >= 1 && ccw(hull[r - 2], hull[r - 1], p[i]) >= 0)
            r--;
        hull[r++] = p[i];
    }
    return l == 1 ? 1 : r - 1;
}
```

Convex hull

Many other algorithms exist

Convex hull

Many other algorithms exist

- Gift wrapping aka Jarvis march.

Convex hull

Many other algorithms exist

- Gift wrapping aka Jarvis march.
- Quick hull, similar idea to quicksort.

Convex hull

Many other algorithms exist

- Gift wrapping aka Jarvis march.
- Quick hull, similar idea to quicksort.
- Divide and conquer.

Convex hull

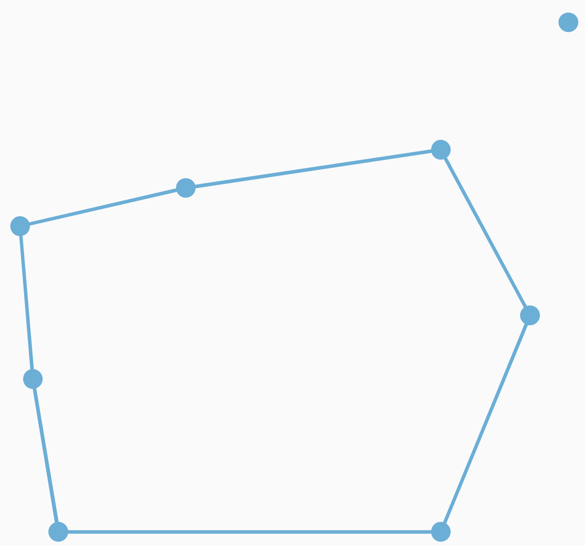
Many other algorithms exist

- Gift wrapping aka Jarvis march.
- Quick hull, similar idea to quicksort.
- Divide and conquer.

Some can be extended to three dimensions, or higher.

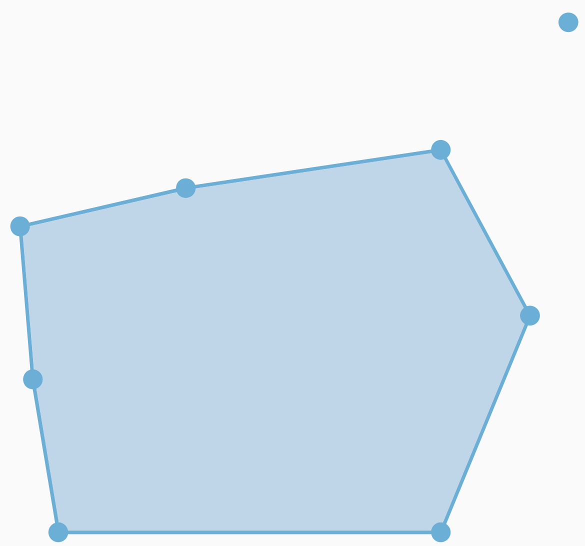
Point in convex polygon

Simple algorithm to check if a point is in a convex polygon.



Point in convex polygon

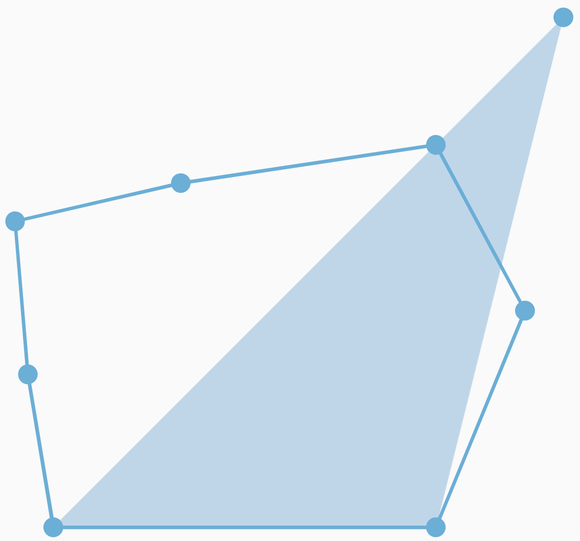
Simple algorithm to check if a point is in a convex polygon.



- We start by calculating the area of the polygon.

Point in convex polygon

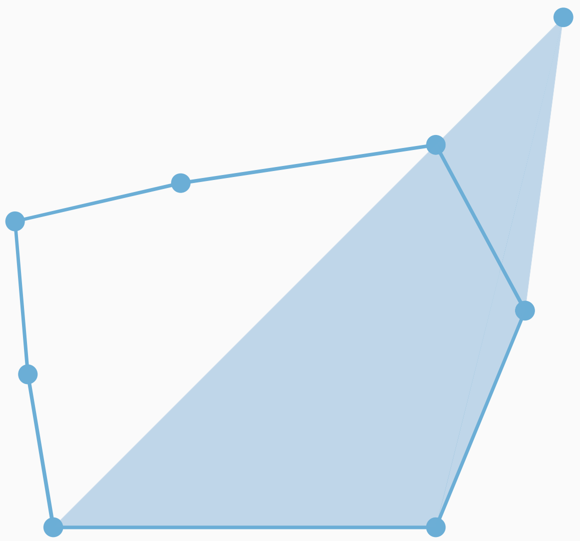
Simple algorithm to check if a point is in a convex polygon.



- We start by calculating the area of the polygon.
- To check if our point is contained in the polygon we sum up the area of the triangles formed the point and every two adjacent points.

Point in convex polygon

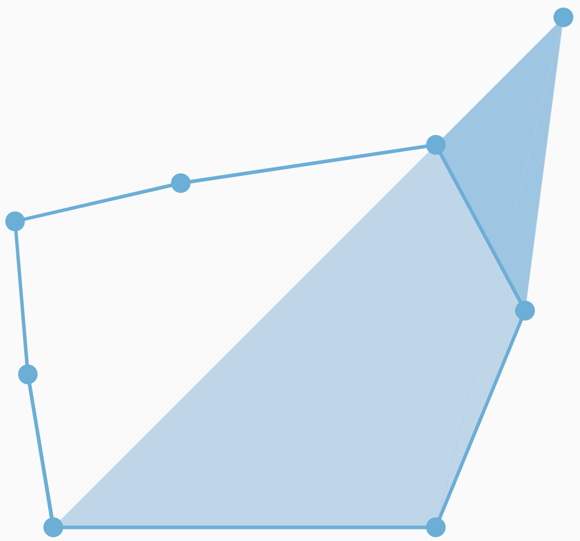
Simple algorithm to check if a point is in a convex polygon.



- We start by calculating the area of the polygon.
- To check if our point is contained in the polygon we sum up the area of the triangles formed the point and every two adjacent points.

Point in convex polygon

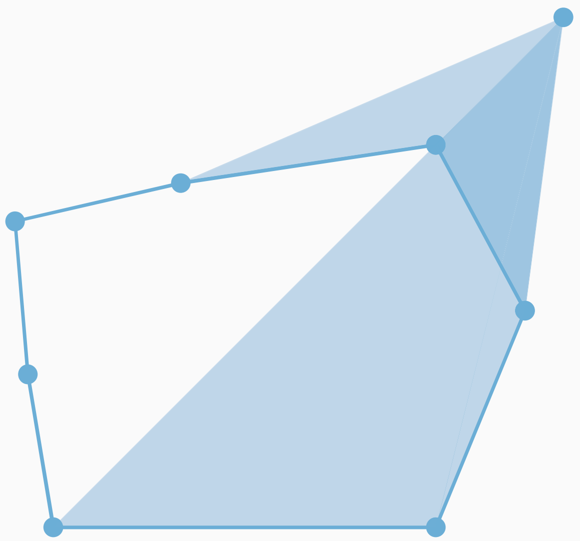
Simple algorithm to check if a point is in a convex polygon.



- We start by calculating the area of the polygon.
- To check if our point is contained in the polygon we sum up the area of the triangles formed the point and every two adjacent points.

Point in convex polygon

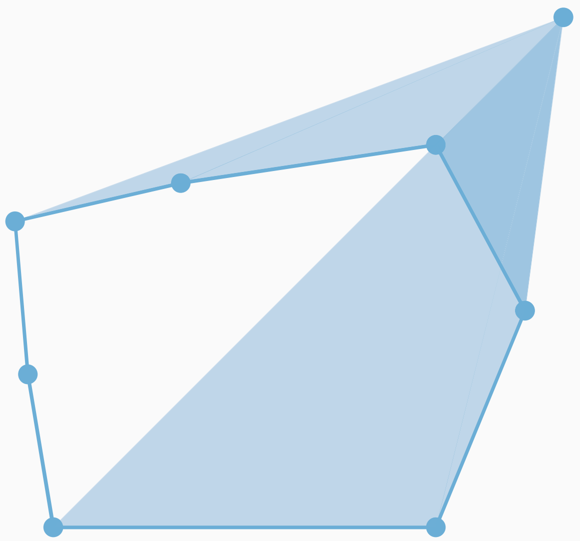
Simple algorithm to check if a point is in a convex polygon.



- We start by calculating the area of the polygon.
- To check if our point is contained in the polygon we sum up the area of the triangles formed the point and every two adjacent points.

Point in convex polygon

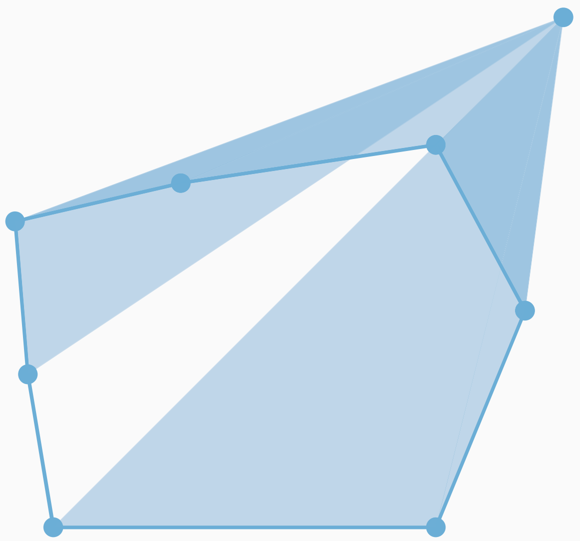
Simple algorithm to check if a point is in a convex polygon.



- We start by calculating the area of the polygon.
- To check if our point is contained in the polygon we sum up the area of the triangles formed the point and every two adjacent points.

Point in convex polygon

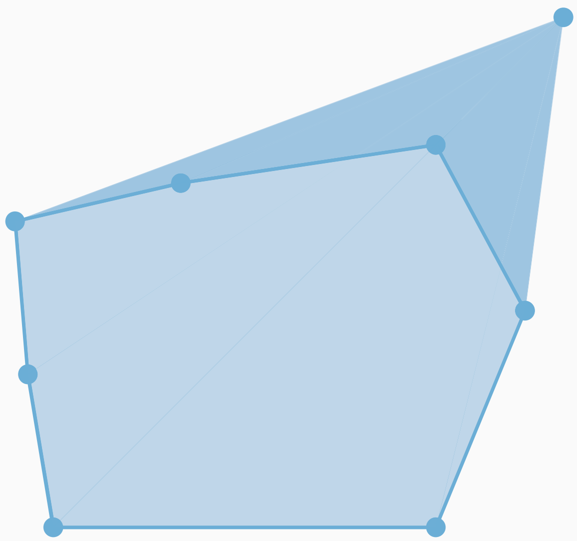
Simple algorithm to check if a point is in a convex polygon.



- We start by calculating the area of the polygon.
- To check if our point is contained in the polygon we sum up the area of the triangles formed the point and every two adjacent points.

Point in convex polygon

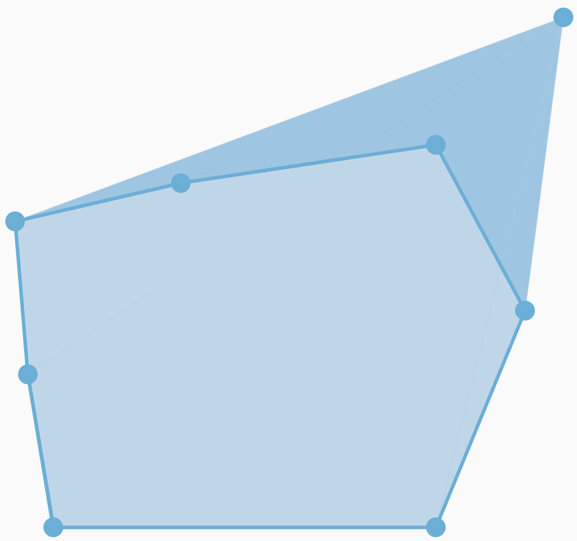
Simple algorithm to check if a point is in a convex polygon.



- We start by calculating the area of the polygon.
- To check if our point is contained in the polygon we sum up the area of the triangles formed the point and every two adjacent points.

Point in convex polygon

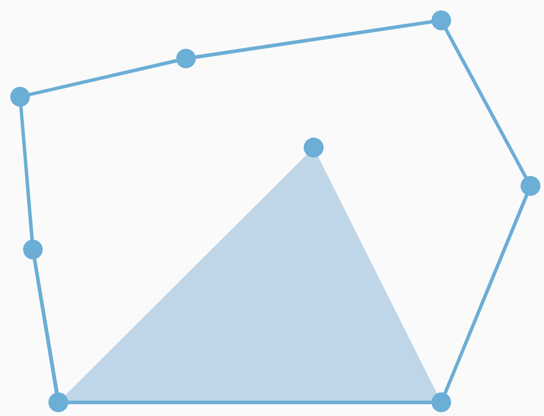
Simple algorithm to check if a point is in a convex polygon.



- We start by calculating the area of the polygon.
- To check if our point is contained in the polygon we sum up the area of the triangles formed the point and every two adjacent points.
- The total area of the triangles is equal to the area of the polygon iff the point is inside the polygon.

Point in convex polygon

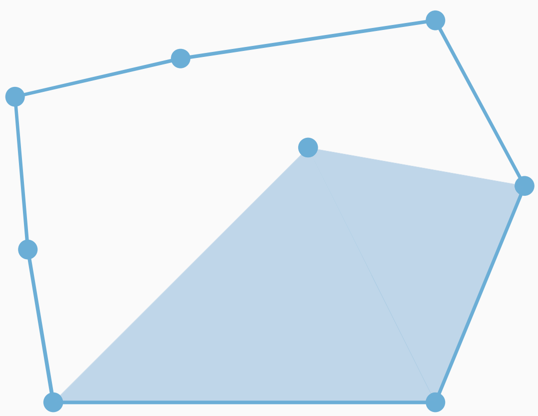
Simple algorithm to check if a point is in a convex polygon.



- We start by calculating the area of the polygon.
- To check if our point is contained in the polygon we sum up the area of the triangles formed the point and every two adjacent points.
- The total area of the triangles is equal to the area of the polygon iff the point is inside the polygon.

Point in convex polygon

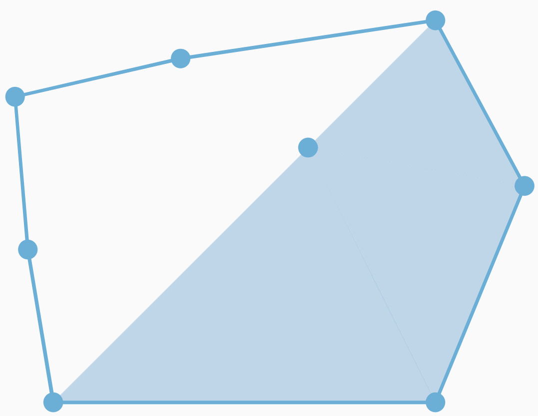
Simple algorithm to check if a point is in a convex polygon.



- We start by calculating the area of the polygon.
- To check if our point is contained in the polygon we sum up the area of the triangles formed the point and every two adjacent points.
- The total area of the triangles is equal to the area of the polygon iff the point is inside the polygon.

Point in convex polygon

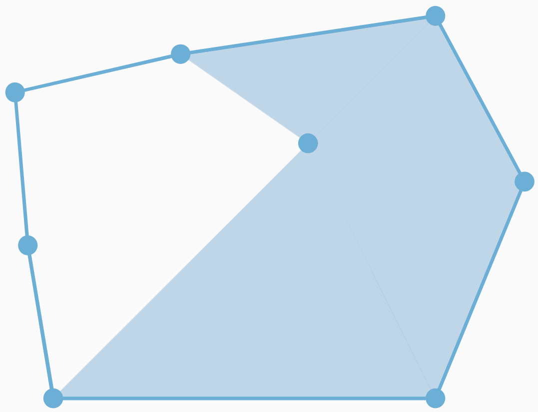
Simple algorithm to check if a point is in a convex polygon.



- We start by calculating the area of the polygon.
- To check if our point is contained in the polygon we sum up the area of the triangles formed the point and every two adjacent points.
- The total area of the triangles is equal to the area of the polygon iff the point is inside the polygon.

Point in convex polygon

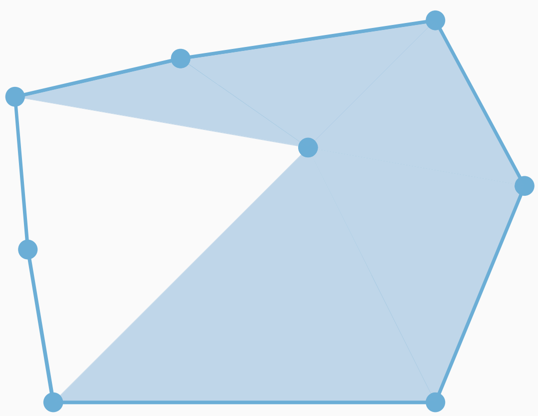
Simple algorithm to check if a point is in a convex polygon.



- We start by calculating the area of the polygon.
- To check if our point is contained in the polygon we sum up the area of the triangles formed the point and every two adjacent points.
- The total area of the triangles is equal to the area of the polygon iff the point is inside the polygon.

Point in convex polygon

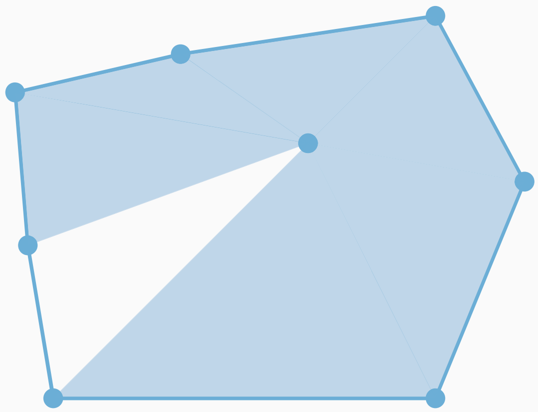
Simple algorithm to check if a point is in a convex polygon.



- We start by calculating the area of the polygon.
- To check if our point is contained in the polygon we sum up the area of the triangles formed the point and every two adjacent points.
- The total area of the triangles is equal to the area of the polygon iff the point is inside the polygon.

Point in convex polygon

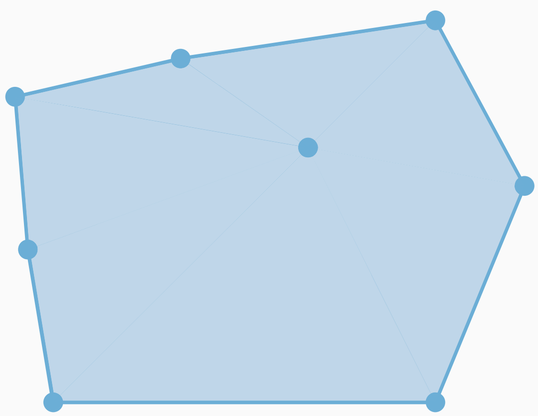
Simple algorithm to check if a point is in a convex polygon.



- We start by calculating the area of the polygon.
- To check if our point is contained in the polygon we sum up the area of the triangles formed the point and every two adjacent points.
- The total area of the triangles is equal to the area of the polygon iff the point is inside the polygon.

Point in convex polygon

Simple algorithm to check if a point is in a convex polygon.



- We start by calculating the area of the polygon.
- To check if our point is contained in the polygon we sum up the area of the triangles formed the point and every two adjacent points.
- The total area of the triangles is equal to the area of the polygon iff the point is inside the polygon.

Point in concave polygon

How about non convex polygon?

Point in concave polygon

How about non convex polygon?

- The *even-odd rule* algorithm.

Point in concave polygon

How about non convex polygon?

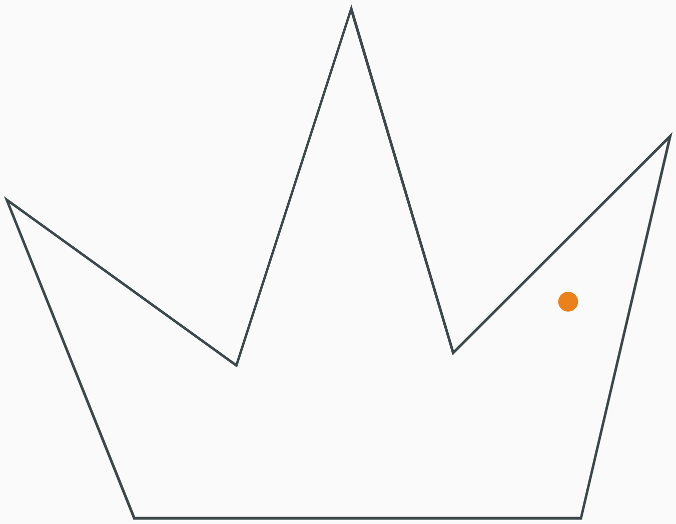
- The *even-odd rule* algorithm.
- We examine a ray passing through the polygon to the point.

Point in concave polygon

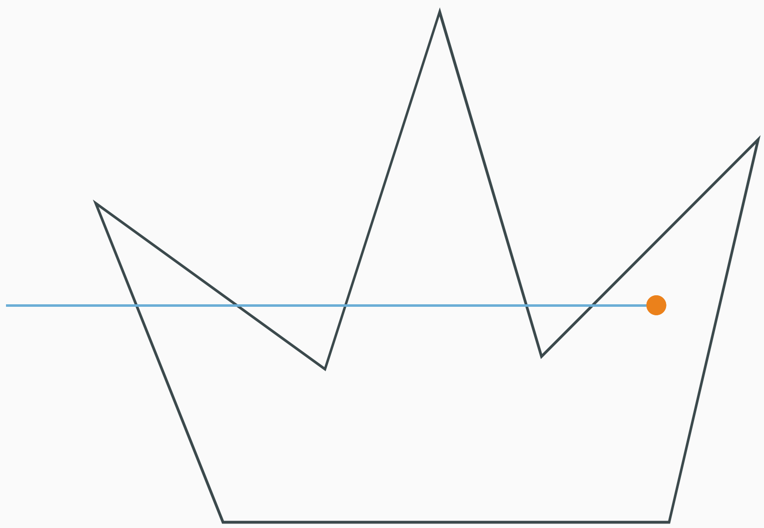
How about non convex polygon?

- The *even-odd rule* algorithm.
- We examine a ray passing through the polygon to the point.
- If the ray crosses the boundary of the polygon, then it alternately goes from outside to inside, and outside to inside.

Point in concave polygon

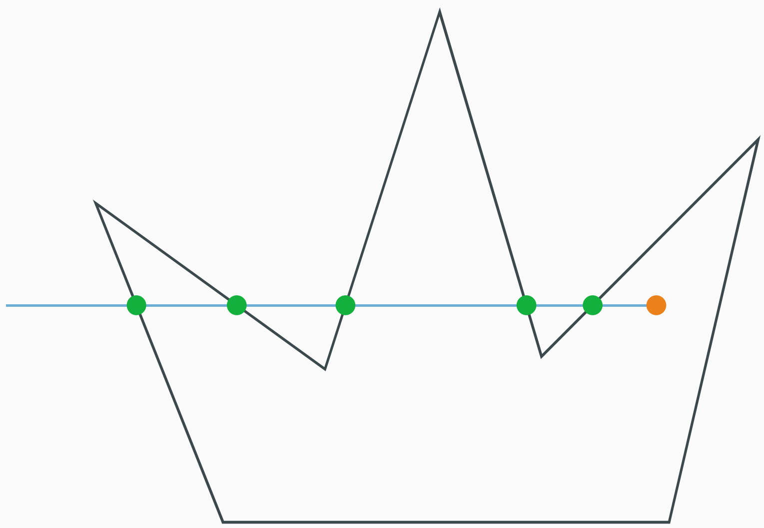


Point in concave polygon



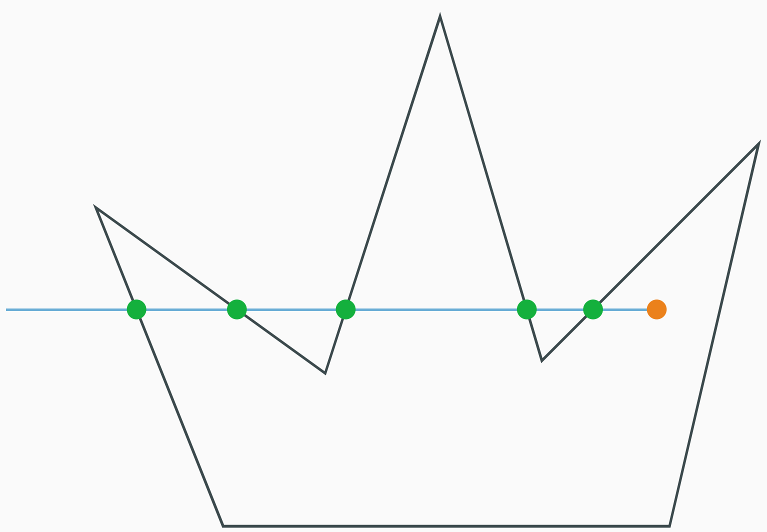
- Ray from the outside of the polygon to the point.

Point in concave polygon



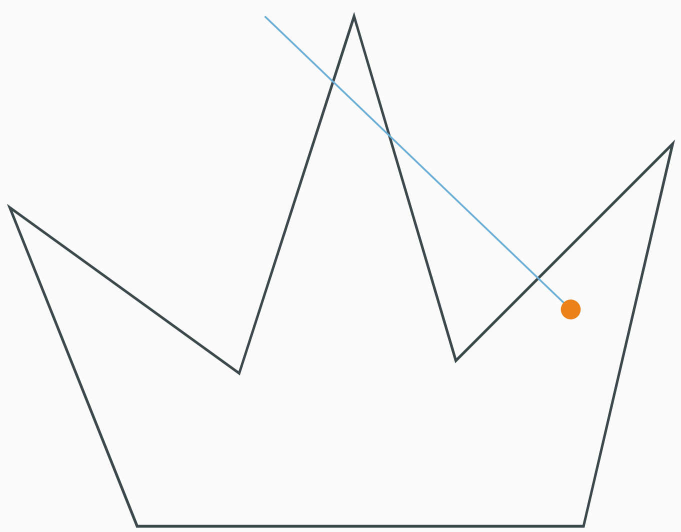
- Ray from the outside of the polygon to the point.
- Count the number of intersection points.

Point in concave polygon



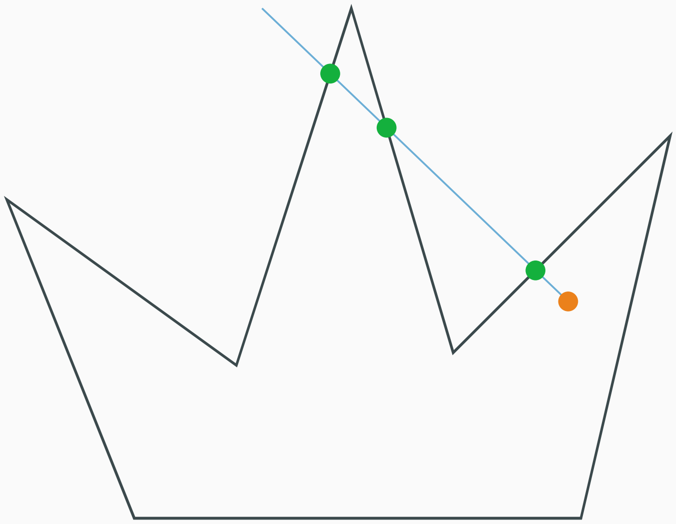
- Ray from the outside of the polygon to the point.
- Count the number of intersection points.
- If odd, then the point is inside the polygon.
- If even, then the point is outside the polygon.

Point in concave polygon



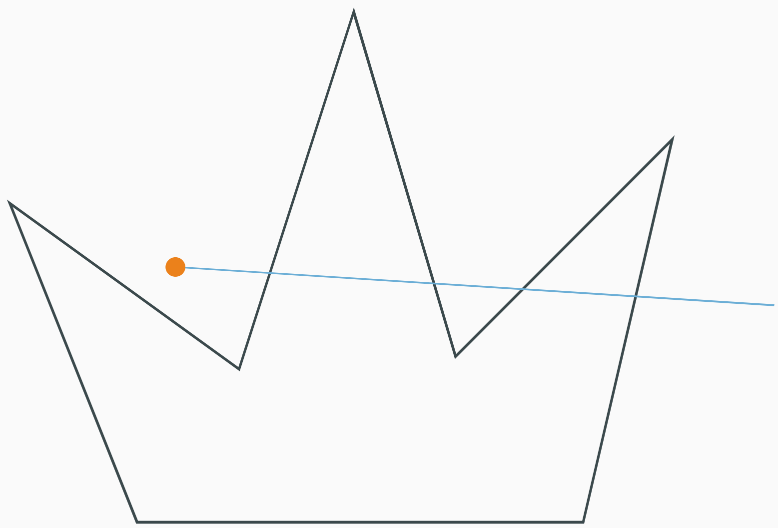
- Ray from the outside of the polygon to the point.
- Count the number of intersection points.
- If odd, then the point is inside the polygon.
- If even, then the point is outside the polygon.
- Does not matter which ray we pick.

Point in concave polygon



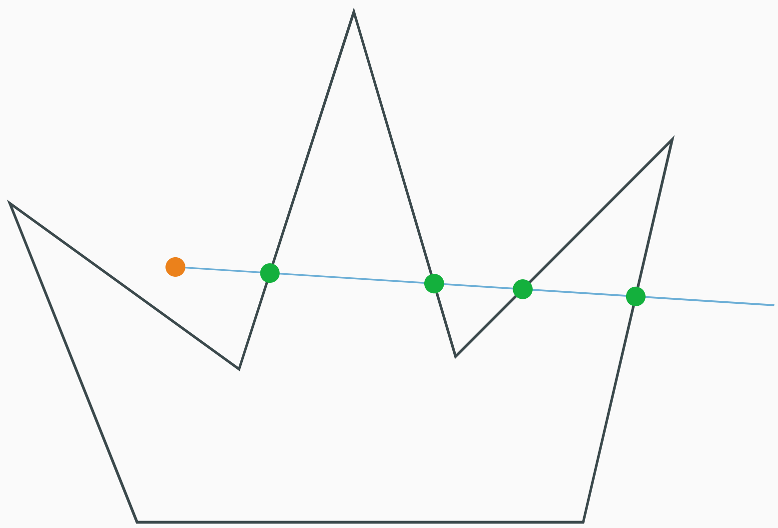
- Ray from the outside of the polygon to the point.
- Count the number of intersection points.
- If odd, then the point is inside the polygon.
- If even, then the point is outside the polygon.
- Does not matter which ray we pick.

Point in concave polygon



- Ray from the outside of the polygon to the point.
- Count the number of intersection points.
- If odd, then the point is inside the polygon.
- If even, then the point is outside the polygon.
- Does not matter which ray we pick.

Point in concave polygon



- Ray from the outside of the polygon to the point.
- Count the number of intersection points.
- If odd, then the point is inside the polygon.
- If even, then the point is outside the polygon.
- Does not matter which ray we pick.

Closest pair of points

Given a set of points, we want to find the pair of points with the smallest distance between them.

Divide and conquer algorithm;