Computational Geometry

Jordan Arenstein

February 7, 2018

Vectors

Vectors

- a vector is a quantity with direction and magnitude
- a vector lists directional components

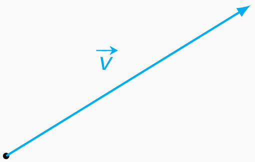
Representing Vectors

$$\vec{v} = \begin{bmatrix} x \\ y \end{bmatrix}$$

•

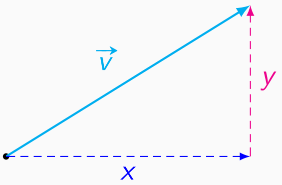
Representing Vectors

$$\vec{v} = \begin{bmatrix} x \\ y \end{bmatrix}$$



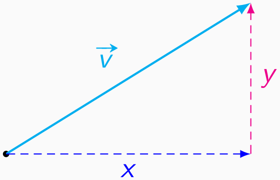
Representing Vectors

$$\vec{v} = \begin{bmatrix} x \\ y \end{bmatrix}$$



Representing Vectors

$$\vec{v} = \begin{bmatrix} x \\ y \end{bmatrix}$$




```
struct vec {
  int x, y;
};
vec v = {x, y};
```

Manipulating Vectors

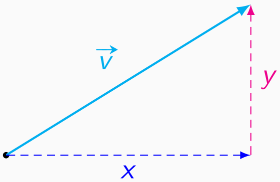
Magnitude

$$|\vec{v}|$$
$$= \sqrt{x^2 + y^2}$$

Magnitude

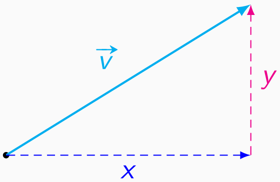
$$|\overrightarrow{v}|$$

$$= \sqrt{x^2 + y^2}$$



Magnitude

$$|\vec{v}| = \sqrt{x^2 + y^2}$$




```
int len() {
  return sqrt(x*x + y*y);
}
```

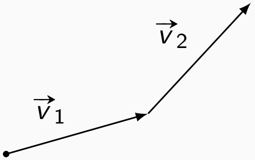
Vector Addition

$$\vec{v}_1 + \vec{v}_2 = \begin{bmatrix} x_1 + x_2 \\ y_1 + y_2 \end{bmatrix}$$

-

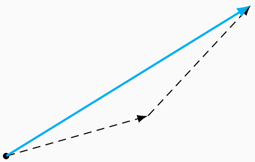
Vector Addition

$$\vec{v}_1 + \vec{v}_2 = \begin{bmatrix} x_1 + x_2 \\ y_1 + y_2 \end{bmatrix}$$



Vector Addition

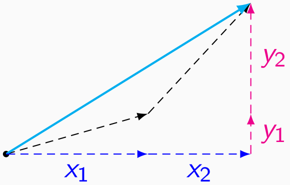
$$\vec{v}_1 + \vec{v}_2 = \begin{bmatrix} x_1 + x_2 \\ y_1 + y_2 \end{bmatrix}$$



Vector Addition

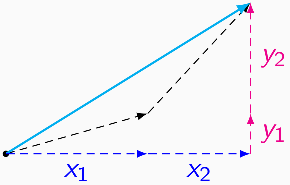
$$\vec{v}_1 + \vec{v}_2$$

$$= \begin{bmatrix} x_1 + x_2 \\ y_1 + y_2 \end{bmatrix}$$



Vector Addition

$$\vec{v}_1 + \vec{v}_2$$
$$= \begin{bmatrix} x_1 + x_2 \\ y_1 + y_2 \end{bmatrix}$$




```
vec add(vec v) {
  return {x+v.x, y+v.y};
}
```

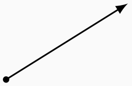
Scalar Multiplication

$$s\vec{v} = \begin{bmatrix} sx \\ sy \end{bmatrix}$$

-

Scalar Multiplication

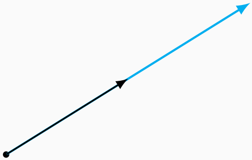
$$s\vec{v} = \begin{bmatrix} sx \\ sy \end{bmatrix}$$



Scalar Multiplication

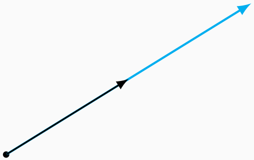
$$s\vec{v}$$

$$= \begin{bmatrix} sx \\ sy \end{bmatrix}$$



Scalar Multiplication

$$s\vec{v} = \begin{bmatrix} sx \\ sy \end{bmatrix}$$



```
vec scale(int s) {
  return vec{s*x, s*y};
}
```

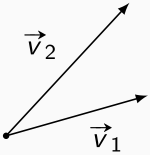
Dot Product

$$\vec{v}_1 \cdot \vec{v}_2$$
$$= |\vec{v}_1|\,|\vec{v}_2| \cos\theta$$
$$= x_1 x_2 + y_1 y_2$$

•

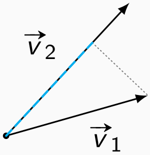
Dot Product

$$\vec{v}_1 \cdot \vec{v}_2$$
$$= |\vec{v}_1|\,|\vec{v}_2| \cos\theta$$
$$= x_1 x_2 + y_1 y_2$$



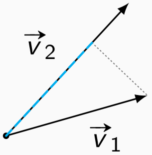
Dot Product

$$\vec{v}_1 \cdot \vec{v}_2$$
$$= |\vec{v}_1|\,|\vec{v}_2| \cos\theta$$
$$= x_1 x_2 + y_1 y_2$$



Dot Product

$$\vec{v}_1 \cdot \vec{v}_2$$
$$= |\vec{v}_1|\,|\vec{v}_2| \cos\theta$$
$$= x_1 x_2 + y_1 y_2$$




```
int dot(vec v) {
  return x*v.x + y*v.x;
}
```

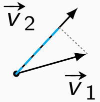
Dot Product Uses

the dot product describes direction

Dot Product Uses

the dot product describes direction

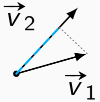
- $\vec{v}_1 \cdot \vec{v}_2 > 0$: $\vec{v}_1$ and $\vec{v}_2$ are directed towards the same half



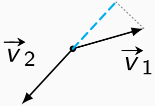
Dot Product Uses

the dot product describes direction

- $\vec{v}_1 \cdot \vec{v}_2 > 0$: $\vec{v}_1$ and $\vec{v}_2$ are directed towards the same half



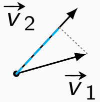
- $\vec{v}_1 \cdot \vec{v}_2 < 0$: $\vec{v}_1$ and $\vec{v}_2$ are directed towards opposite halves



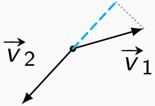
Dot Product Uses

the dot product describes direction

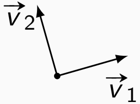
- $\vec{v}_1 \cdot \vec{v}_2 > 0$: $\vec{v}_1$ and $\vec{v}_2$ are directed towards the same half



- $\vec{v}_1 \cdot \vec{v}_2 < 0$: $\vec{v}_1$ and $\vec{v}_2$ are directed towards opposite halves



- $\vec{v}_1 \cdot \vec{v}_2 = 0$: $\vec{v}_1$ and $\vec{v}_2$ are perpendicular



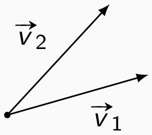
Cross Product

$$
\begin{aligned}
&\vec{v}_1 \times \vec{v}_2 \\
&= |\vec{v}_1|\,|\vec{v}_2| \sin\theta \\
&= x_1 y_2 - y_1 x_2
\end{aligned}
$$

-

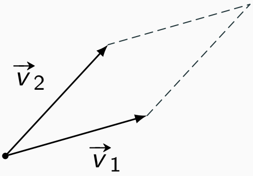
Cross Product

$$\vec{v}_1 \times \vec{v}_2$$
$$= |\vec{v}_1|\,|\vec{v}_2| \sin\theta$$
$$= x_1 y_2 - y_1 x_2$$



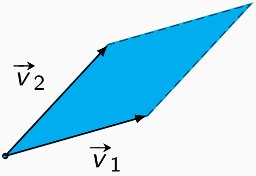
Cross Product

$$\vec{v}_1 \times \vec{v}_2$$
$$= |\vec{v}_1|\,|\vec{v}_2| \sin\theta$$
$$= x_1 y_2 - y_1 x_2$$



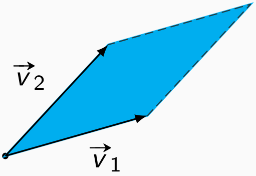
Cross Product

$$\vec{v}_1 \times \vec{v}_2$$
$$= |\vec{v}_1|\,|\vec{v}_2| \sin\theta$$
$$= x_1 y_2 - y_1 x_2$$



Cross Product

$$\vec{v}_1 \times \vec{v}_2$$
$$= |\vec{v}_1|\,|\vec{v}_2| \sin\theta$$
$$= x_1 y_2 - y_1 x_2$$



```
int cross(vec v) {
  return x*v.y - y*v.x;
}
```

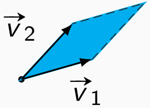
Cross Product Uses

the cross product describes orientation

Cross Product Uses

the cross product describes orientation

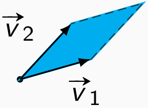
- $\vec{v}_1 \times \vec{v}_2 > 0$: $\vec{v}_1$ is clockwise from $\vec{v}_2$



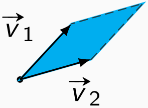
Cross Product Uses

the cross product describes orientation

- $\vec{v}_1 \times \vec{v}_2 > 0$: $\vec{v}_1$ is clockwise from $\vec{v}_2$



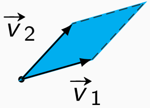
- $\vec{v}_1 \times \vec{v}_2 < 0$: $\vec{v}_1$ is anti-clockwise from $\vec{v}_2$



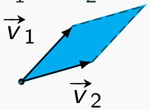
Cross Product Uses

the cross product describes orientation

- $\vec{v}_1 \times \vec{v}_2 > 0$: $\vec{v}_1$ is clockwise from $\vec{v}_2$



- $\vec{v}_1 \times \vec{v}_2 < 0$: $\vec{v}_1$ is anti-clockwise from $\vec{v}_2$



- $\vec{v}_1 \times \vec{v}_2 = 0$: $\vec{v}_1$ and $\vec{v}_2$ are collinear

Lines

Lines

lines can be represented as a pair of vectors

- $\vec{p} = \vec{b} + s\vec{m}$
- $\vec{b}$: a vector for position
- $\vec{m}$: a vector for slope
- the vectors must be standardised in order to check for equality

Representing Line Segments

$$\overline{AB}$$

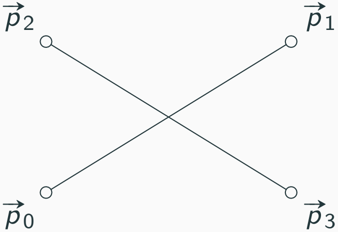
- a line segment can be represented by a pair of vectors
- $\vec{p}_1 - \vec{p}_0$ gives a vector representing the line segments' length and direction

Intersecting Line Segments

the cross product allows efficient checks for intersecting line segments

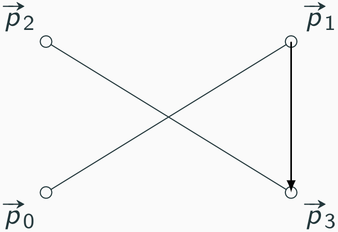
Intersecting Line Segments

the cross product allows efficient checks for intersecting line segments



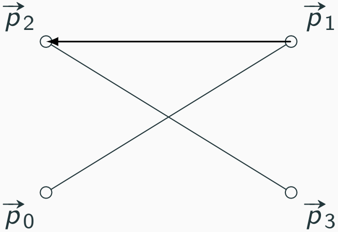
Intersecting Line Segments

the cross product allows efficient checks for intersecting line segments



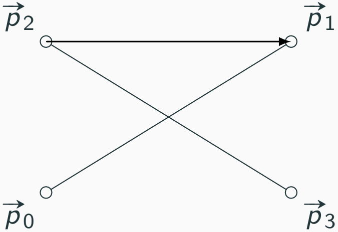
Intersecting Line Segments

the cross product allows efficient checks for intersecting line segments



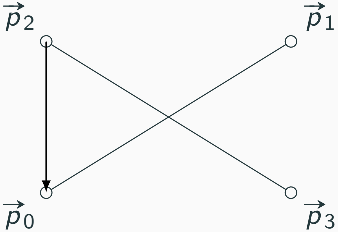
Intersecting Line Segments

the cross product allows efficient checks for intersecting line segments



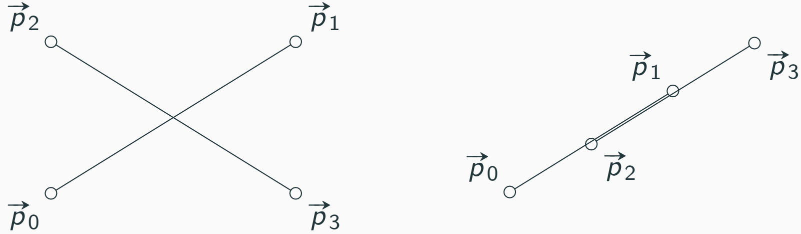
Intersecting Line Segments

the cross product allows efficient checks for intersecting line segments



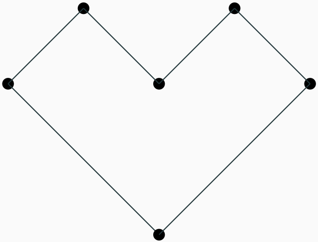
Intersecting Line Segments

the cross product allows efficient checks for intersecting line segments



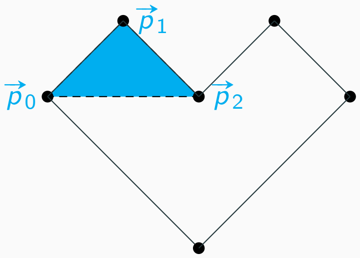
Polygons

calculating the area of a polygon can be simplified by triangulating the polygon



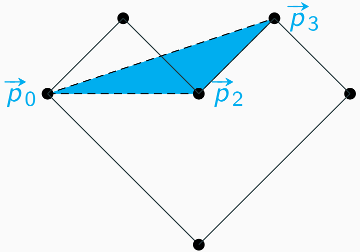
Area of Polygons

calculating the area of a polygon can be simplified by triangulating the polygon



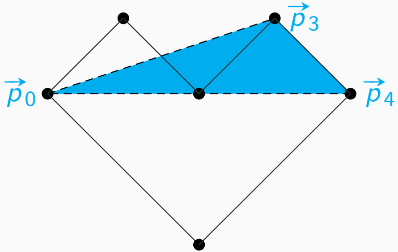
Area of Polygons

calculating the area of a polygon can be simplified by triangulating the polygon



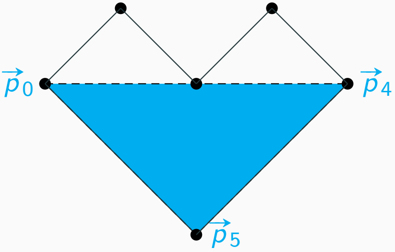
Area of Polygons

calculating the area of a polygon can be simplified by triangulating the polygon



Area of Polygons

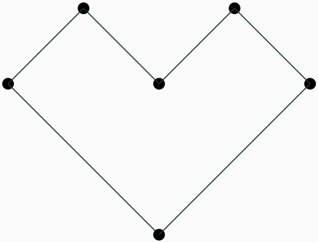
calculating the area of a polygon can be simplified by triangulating the polygon



Area of Polygons

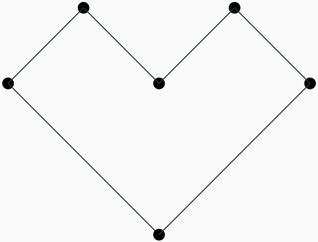
calculating the area of a polygon can be simplified by triangulating the polygon

the cross product is used to determine the area



Area of Polygons

calculating the area of a polygon can be simplified by triangulating the polygon

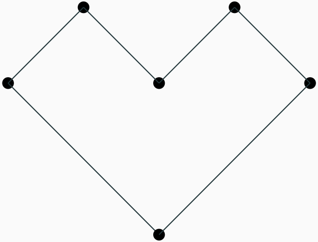


the cross product is used to determine the area

- it returns the area of the parallelogram with two vectors common to each triangle

Area of Polygons

calculating the area of a polygon can be simplified by triangulating the polygon

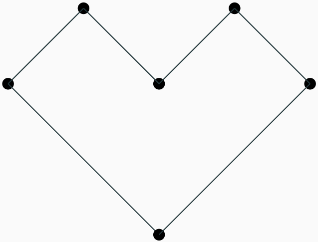


the cross product is used to determine the area

- it returns the area of the parallelogram with two vectors common to each triangle
- the area is signed depending on orientation

Area of Polygons

calculating the area of a polygon can be simplified by triangulating the polygon

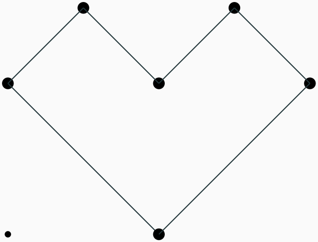


the cross product is used to determine the area

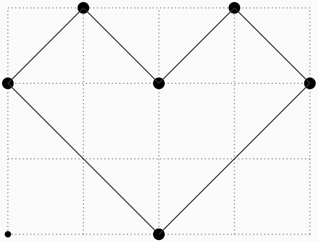
- it returns the area of the parallelogram with two vectors common to each triangle
- the area is signed depending on orientation

$$2A(\Omega) = \left| \sum_{i=1}^{n-2} (\vec{p}_i - \vec{p}_0) \times (\vec{p}_{i+1} - \vec{p}_0) \right|$$

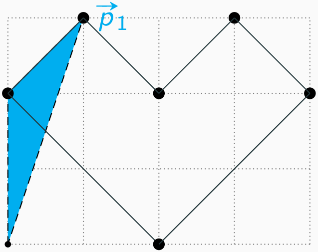
Optimising Area of Polygons



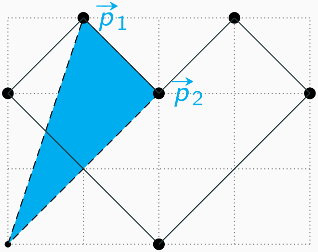
Optimising Area of Polygons



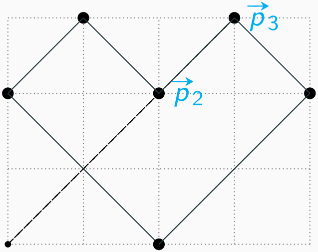
Optimising Area of Polygons



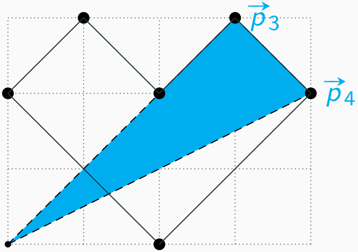
Optimising Area of Polygons



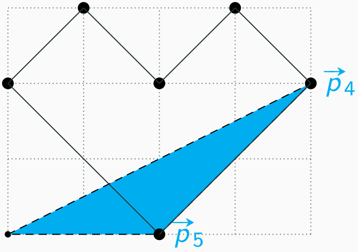
Optimising Area of Polygons



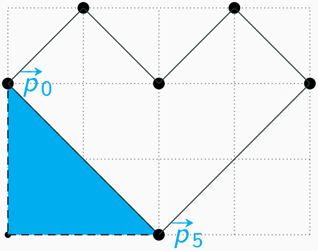
Optimising Area of Polygons



Optimising Area of Polygons

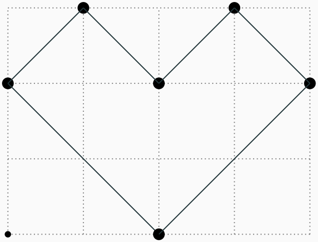


Optimising Area of Polygons



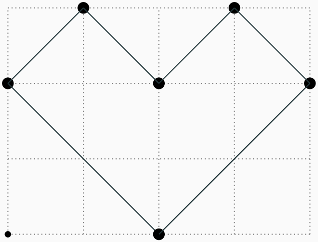
Optimising Area of Polygons

$$2A(\Omega) = \left| \sum_{i=0}^{n-1} \vec{p}_i \times \vec{p}_{i+1} \right|$$



Optimising Area of Polygons

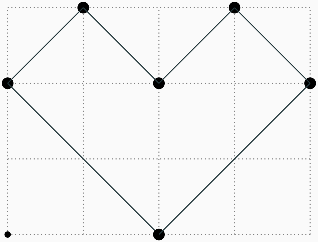
$$2A(\Omega) = \left| \sum_{i=0}^{n-1} \vec{p}_i \times \vec{p}_{i+1} \right|$$



$$2A(\Omega) = \left| \sum_{i=0}^{n-1} x_i y_{i+1} - x_{i+1} y_i \right|$$

Optimising Area of Polygons

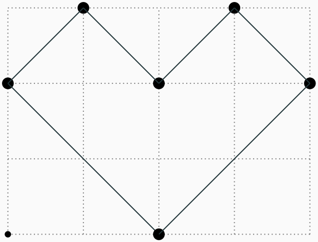
$$2A(\Omega) = \left| \sum_{i=0}^{n-1} \vec{p}_i \times \vec{p}_{i+1} \right|$$



$$2A(\Omega) = \left| \sum_{i=0}^{n-1} x_i y_{i+1} - x_{i+1} y_i \right|$$

$$2A(\Omega) = \left| \sum_{i=0}^{n-1} (x_i + x_{i+1})(y_{i+1} - y_i) \right|$$

Optimising Area of Polygons



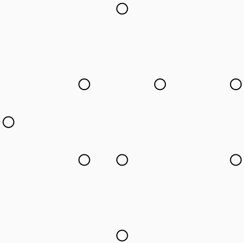
$$2A(\Omega) = \left| \sum_{i=0}^{n-1} \vec{p}_i \times \vec{p}_{i+1} \right|$$

$$2A(\Omega) = \left| \sum_{i=0}^{n-1} x_i y_{i+1} - x_{i+1} y_i \right|$$

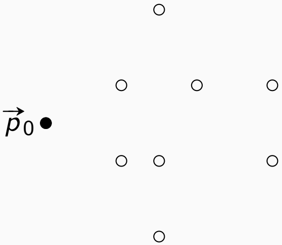
$$2A(\Omega) = \left| \sum_{i=0}^{n-1} (x_i + x_{i+1})(y_{i+1} - y_i) \right|$$

$$2A(\Omega) = \left| \sum_{i=0}^{n-1} x_i (y_{i+1} - y_{i-1}) \right|$$

Constructing Convex Polygons - Graham Scan

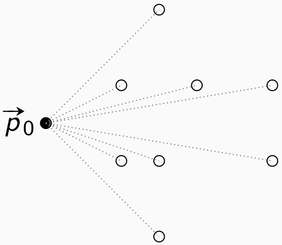


Constructing Convex Polygons - Graham Scan



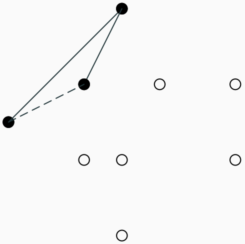
- pick a point $\vec{p}_0$ that is definitely on the convex hull

Constructing Convex Polygons - Graham Scan



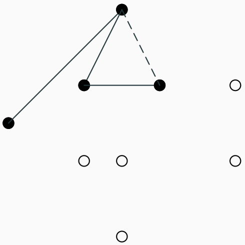
- pick a point $\vec{p}_0$ that is definitely on the convex hull
- sort the remaining points according to their orientation from $\vec{p}_0$

Constructing Convex Polygons - Graham Scan



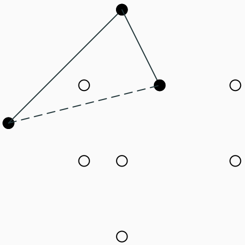
- pick a point $\vec{p}_0$ that is definitely on the convex hull
- sort the remaining points according to their orientation from $\vec{p}_0$
- push $\vec{p}_0$ to a stack H

Constructing Convex Polygons - Graham Scan



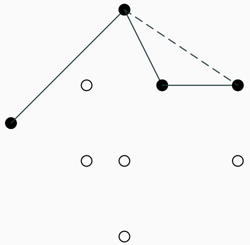
- pick a point $\vec{p}_0$ that is definitely on the convex hull
- sort the remaining points according to their orientation from $\vec{p}_0$
- push $\vec{p}_0$ to a stack H
- for each point $\vec{p}_i$:
    - while $\vec{p}_i$ is left of the line formed by the previous two points on H, pop the last point on H
    - push $\vec{p}_i$ to H

Constructing Convex Polygons - Graham Scan



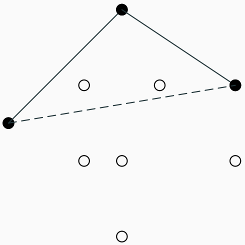
- pick a point $\vec{p}_0$ that is definitely on the convex hull
- sort the remaining points according to their orientation from $\vec{p}_0$
- push $\vec{p}_0$ to a stack H
- for each point $\vec{p}_i$:
    - while $\vec{p}_i$ is left of the line formed by the previous two points on H, pop the last point on H
    - push $\vec{p}_i$ to H

Constructing Convex Polygons - Graham Scan



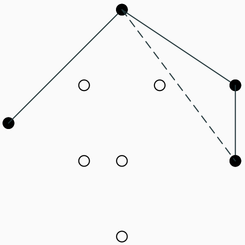
- pick a point $\vec{p}_0$ that is definitely on the convex hull
- sort the remaining points according to their orientation from $\vec{p}_0$
- push $\vec{p}_0$ to a stack H
- for each point $\vec{p}_i$:
    - while $\vec{p}_i$ is left of the line formed by the previous two points on H, pop the last point on H
    - push $\vec{p}_i$ to H

Constructing Convex Polygons - Graham Scan



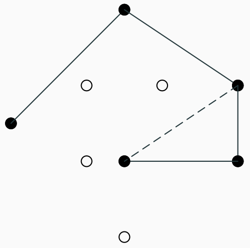
- pick a point $\vec{p}_0$ that is definitely on the convex hull
- sort the remaining points according to their orientation from $\vec{p}_0$
- push $\vec{p}_0$ to a stack H
- for each point $\vec{p}_i$:
    - while $\vec{p}_i$ is left of the line formed by the previous two points on H, pop the last point on H
    - push $\vec{p}_i$ to H

Constructing Convex Polygons - Graham Scan



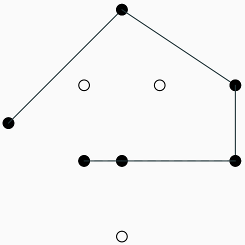
- pick a point $\vec{p}_0$ that is definitely on the convex hull
- sort the remaining points according to their orientation from $\vec{p}_0$
- push $\vec{p}_0$ to a stack H
- for each point $\vec{p}_i$:
    - while $\vec{p}_i$ is left of the line formed by the previous two points on H, pop the last point on H
    - push $\vec{p}_i$ to H

Constructing Convex Polygons - Graham Scan



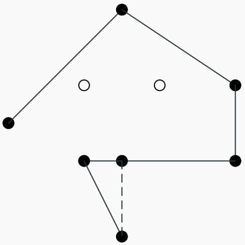
- pick a point $\vec{p}_0$ that is definitely on the convex hull
- sort the remaining points according to their orientation from $\vec{p}_0$
- push $\vec{p}_0$ to a stack H
- for each point $\vec{p}_i$:
    - while $\vec{p}_i$ is left of the line formed by the previous two points on H, pop the last point on H
    - push $\vec{p}_i$ to H

Constructing Convex Polygons - Graham Scan



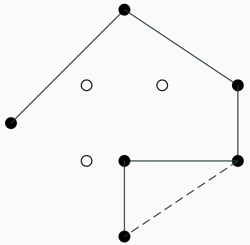
- pick a point $\vec{p}_0$ that is definitely on the convex hull
- sort the remaining points according to their orientation from $\vec{p}_0$
- push $\vec{p}_0$ to a stack H
- for each point $\vec{p}_i$:
    - while $\vec{p}_i$ is left of the line formed by the previous two points on H, pop the last point on H
    - push $\vec{p}_i$ to H

Constructing Convex Polygons - Graham Scan



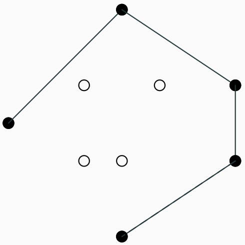
- pick a point $\vec{p}_0$ that is definitely on the convex hull
- sort the remaining points according to their orientation from $\vec{p}_0$
- push $\vec{p}_0$ to a stack H
- for each point $\vec{p}_i$:
    - while $\vec{p}_i$ is left of the line formed by the previous two points on H, pop the last point on H
    - push $\vec{p}_i$ to H

Constructing Convex Polygons - Graham Scan



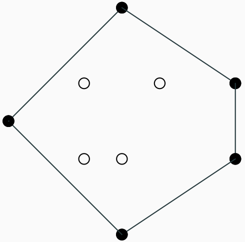
- pick a point $\vec{p}_0$ that is definitely on the convex hull
- sort the remaining points according to their orientation from $\vec{p}_0$
- push $\vec{p}_0$ to a stack H
- for each point $\vec{p}_i$:
    - while $\vec{p}_i$ is left of the line formed by the previous two points on H, pop the last point on H
    - push $\vec{p}_i$ to H

Constructing Convex Polygons - Graham Scan



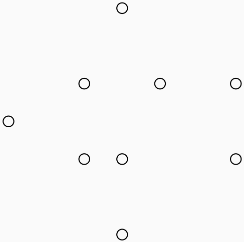
- pick a point $\vec{p}_0$ that is definitely on the convex hull
- sort the remaining points according to their orientation from $\vec{p}_0$
- push $\vec{p}_0$ to a stack H
- for each point $\vec{p}_i$:
    - while $\vec{p}_i$ is left of the line formed by the previous two points on H, pop the last point on H
    - push $\vec{p}_i$ to H

Constructing Convex Polygons - Graham Scan

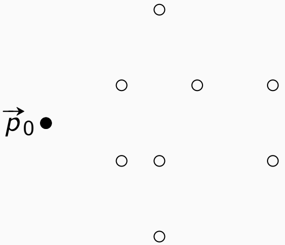


- pick a point $\vec{p}_0$ that is definitely on the convex hull
- sort the remaining points according to their orientation from $\vec{p}_0$
- push $\vec{p}_0$ to a stack H
- for each point $\vec{p}_i$:
    - while $\vec{p}_i$ is left of the line formed by the previous two points on H, pop the last point on H
    - push $\vec{p}_i$ to H

Constructing Convex Polygons - Jarvis' March

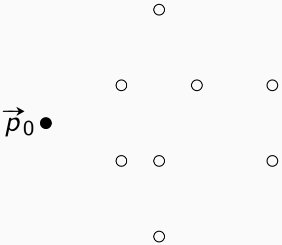


Constructing Convex Polygons - Jarvis' March



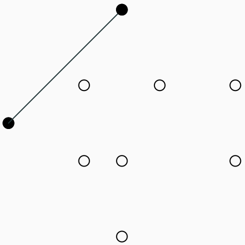
- pick a point $\vec{p}_0$ that is definitely on the convex hull

Constructing Convex Polygons - Jarvis' March



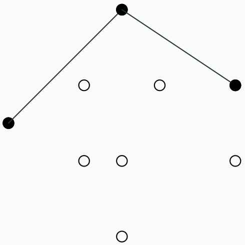
- pick a point $\vec{p}_0$ that is definitely on the convex hull
- append $\vec{p}_0$ to an array H

Constructing Convex Polygons - Jarvis' March



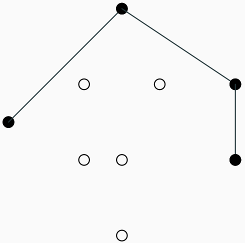
- pick a point $\vec{p}_0$ that is definitely on the convex hull
- append $\vec{p}_0$ to an array H
- append the leftmost point from the last point in H to H until the leftmost point is $\vec{p}_0$

Constructing Convex Polygons - Jarvis' March



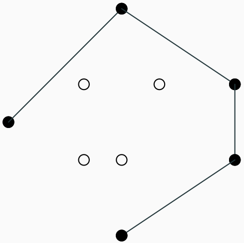
- pick a point $\vec{p}_0$ that is definitely on the convex hull
- append $\vec{p}_0$ to an array H
- append the leftmost point from the last point in H to H until the leftmost point is $\vec{p}_0$

Constructing Convex Polygons - Jarvis' March



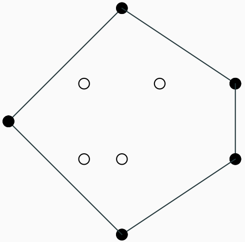
- pick a point $\vec{p}_0$ that is definitely on the convex hull
- append $\vec{p}_0$ to an array H
- append the leftmost point from the last point in H to H until the leftmost point is $\vec{p}_0$

Constructing Convex Polygons - Jarvis' March



- pick a point $\vec{p}_0$ that is definitely on the convex hull
- append $\vec{p}_0$ to an array H
- append the leftmost point from the last point in H to H until the leftmost point is $\vec{p}_0$

Constructing Convex Polygons - Jarvis' March



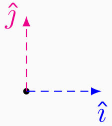
- pick a point $\vec{p}_0$ that is definitely on the convex hull
- append $\vec{p}_0$ to an array H
- append the leftmost point from the last point in H to H until the leftmost point is $\vec{p}_0$

Linear Algebra

Basis Vectors

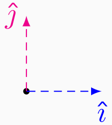
a vector lists directional components: x and y

- $\hat{\imath} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \hat{\jmath} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$



Basis Vectors

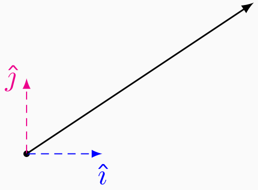
a vector lists directional components: x and y



- $\hat{\imath} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, $\hat{\jmath} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$
- a vector is constructed by scaling and adding basis vectors
- $\begin{bmatrix} x \\ y \end{bmatrix} \rightarrow x\hat{\imath} + y\hat{\jmath}$

Basis Vectors

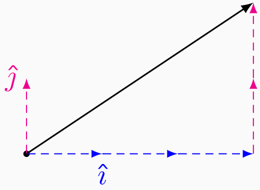
a vector lists directional components: x and y



- $\hat{\imath} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \hat{\jmath} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$
- a vector is constructed by scaling and adding basis vectors
- $\begin{bmatrix} x \\ y \end{bmatrix} \rightarrow x\hat{\imath} + y\hat{\jmath}$

Basis Vectors

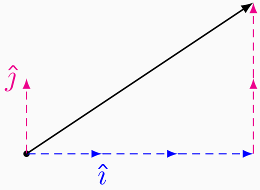
a vector lists directional components: x and y



- $\hat{\imath} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, $\hat{\jmath} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$
- a vector is constructed by scaling and adding basis vectors
- $\begin{bmatrix} x \\ y \end{bmatrix} \to x\hat{\imath} + y\hat{\jmath}$

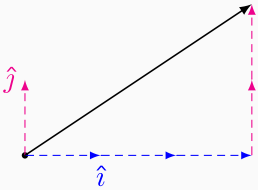
Basis Vectors

a vector lists directional components: x and y



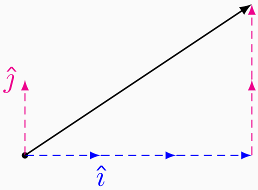
- $\hat{\imath} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \hat{\jmath} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$
- a vector is constructed by scaling and adding basis vectors
- $\begin{bmatrix} x \\ y \end{bmatrix} \rightarrow x\hat{\imath} + y\hat{\jmath}$
- $\hat{\imath}$ and $\hat{\jmath}$ can be grouped in a matrix $\begin{bmatrix} \hat{\imath} & \hat{\jmath} \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

Changing Basis Vectors



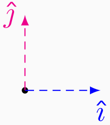
- by changing the basis vectors, any vector in a system can be changed

Changing Basis Vectors



- by changing the basis vectors, any vector in a system can be changed

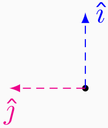
Changing Basis Vectors



- by changing the basis vectors, any vector in a system can be changed
- rotation by 90 degrees

$$\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

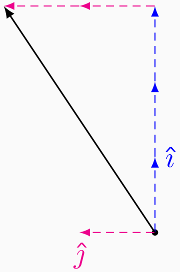
Changing Basis Vectors



- by changing the basis vectors, any vector in a system can be changed
- rotation by 90 degrees

$$\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

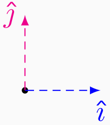
Changing Basis Vectors



- by changing the basis vectors, any vector in a system can be changed
- rotation by 90 degrees

$$\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

Changing Basis Vectors



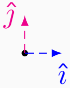
- by changing the basis vectors, any vector in a system can be changed
- rotation by 90 degrees

$$\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

- scaling

$$\begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{bmatrix}$$

Changing Basis Vectors



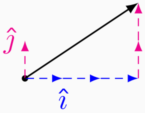
- by changing the basis vectors, any vector in a system can be changed
- rotation by 90 degrees

$$\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

- scaling

$$\begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{bmatrix}$$

Changing Basis Vectors



- by changing the basis vectors, any vector in a system can be changed
- rotation by 90 degrees

$$\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

- scaling

$$\begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{bmatrix}$$

Matrix Multiplication

these transformations are applied by multiplying a vector by a transformation matrix

$$\begin{bmatrix} a & c \\ b & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \to \begin{bmatrix} ax + cy \\ bx + dy \end{bmatrix}$$