Geometry Guide

---

1. Points and Vectors

Points and vectors form the basis of computational geometry. They represent positions and directions in space, respectively.

Definitions:

- A **point** is a location in a coordinate space, usually represented as (x, y) in 2D.
- A **vector** is a quantity with both magnitude and direction, often described as a point with components (x, y).

Vector Operations:

- **Addition**: $(x_1, y_1) + (x_2, y_2) = (x_1 + x_2, y_1 + y_2)$
- **Subtraction**: $(x_1, y_1) - (x_2, y_2) = (x_1 - x_2, y_1 - y_2)$
- **Scalar Multiplication**: $k \times (x, y) = (k \times x, k \times y)$

Code Example:

```cpp
#include <iostream>
#include <cmath>

struct Point {
    double x, y;

    Point operator + (const Point& p) const {
        return {x + p.x, y + p.y};
    }

    Point operator - (const Point& p) const {
        return {x - p.x, y - p.y};
    }

    Point operator * (double k) const {
        return {x * k, y * k};
    }
};

int main() {
    Point p1 = {3, 4}, p2 = {1, 2};
    Point result = p1 + p2;
```

```
    std::cout << "Result: (" << result.x << ", " << result.y << ")\n";
}
```

---

2. Dot Product

The **dot product** measures how aligned two vectors are. It's particularly useful for determining the angle between vectors.

Formula:

Given vectors $\vec{A} = (x_1, y_1)$ and $\vec{B} = (x_2, y_2)$,

$$\vec{A} \cdot \vec{B} = x_1 \times x_2 + y_1 \times y_2$$

- **Properties**:
  - If $\vec{A} \cdot \vec{B} = 0$, $\vec{A}$ and $\vec{B}$ are perpendicular.
  - If $\vec{A} \cdot \vec{B} > 0$, the angle between them is acute.
  - If $\vec{A} \cdot \vec{B} < 0$, the angle is obtuse.

Code Example:

```cpp
double dotProduct(const Point& a, const Point& b) {
    return a.x * b.x + a.y * b.y;
}

int main() {
    Point a = {1, 2}, b = {3, 4};
    std::cout << "Dot Product: " << dotProduct(a, b) << "\n";
}
```

---

3. Cross Product

The **cross product** is used to determine the area of the parallelogram formed by two vectors and to check the relative orientation of points.

Formula:

For vectors $\vec{A} = (x_1, y_1)$ and $\vec{B} = (x_2, y_2)$,

$$\vec{A} \times \vec{B} = x_1 \times y_2 - y_1 \times x_2$$

- **Properties**:
  - If $\vec{A} \times \vec{B} = 0$, the vectors are collinear.
  - If $\vec{A} \times \vec{B} > 0$, $\vec{A}$ is counterclockwise to $\vec{B}$.
  - If $\vec{A} \times \vec{B} < 0$, $\vec{A}$ is clockwise to $\vec{B}$.

Code Example:

cpp

```cpp
double crossProduct(const Point& a, const Point& b) {
    return a.x * b.y - a.y * b.x;
}

int main() {
    Point a = {1, 2}, b = {3, 4};
    std::cout << "Cross Product: " << crossProduct(a, b) << "\n";
}
```

---

4. Distance Between Two Points

The **Euclidean distance** between two points (x_1, y_1) and (x_2, y_2) is given by:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Code Example:

cpp

```cpp
#include <cmath>

double distance(const Point& a, const Point& b) {
    return std::hypot(a.x - b.x, a.y - b.y);
}

int main() {
    Point a = {1, 2}, b = {4, 6};
    std::cout << "Distance: " << distance(a, b) << "\n";
}
```

---

5. Angle Between Two Vectors

To find the angle θ between two vectors, use the dot product:

$$\cos(\theta) = \frac{\vec{A} \cdot \vec{B}}{|\vec{A}| \times |\vec{B}|}$$

where $|\vec{A}| = \sqrt{x_1^2 + y_1^2}$ and $|\vec{B}| = \sqrt{x_2^2 + y_2^2}$.

Code Example:

cpp

```cpp
double angleBetween(const Point& a, const Point& b) {
    double dot = dotProduct(a, b);
    double magA = std::hypot(a.x, a.y);
    double magB = std::hypot(b.x, b.y);
    return std::acos(dot / (magA * magB));
}

int main() {
```

```cpp
    Point a = {1, 0}, b = {0, 1};
    std::cout << "Angle (in radians): " << angleBetween(a, b) << "\n";
}
```

---

6. Line Equation

Given two points, you can form a line equation in the form $ax + by + c = 0$. For points (x_1, y_1) and (x_2, y_2),

$$a = y_2 - y_1, \quad b = x_1 - x_2, \quad c = a \times x_1 + b \times y_1$$

Code Example:

```cpp
struct Line {
    double a, b, c;
};

Line lineFromPoints(const Point& p1, const Point& p2) {
    Line line;
    line.a = p2.y - p1.y;
    line.b = p1.x - p2.x;
    line.c = line.a * p1.x + line.b * p1.y;
    return line;
}

int main() {
    Point p1 = {1, 1}, p2 = {4, 5};
    Line l = lineFromPoints(p1, p2);
    std::cout << "Line equation: " << l.a << "x + " << l.b << "y = " << l.c << "\n";
}
```

---

1. Triangle

A triangle is a polygon with three edges and three vertices.

Properties:

- **Sides**: a, b, and c.
- **Height (h)**: perpendicular distance from the base to the opposite vertex.

- **Area**: $\text{Area} = \frac{1}{2} \times \text{base} \times \text{height}$.
- **Perimeter**: $a + b + c$.

Additional Formulas:

1. **Area using Heron's Formula**:

$$\text{Area} = \sqrt{s \times (s - a) \times (s - b) \times (s - c)}$$

   where $s = \frac{a+b+c}{2}$.

2. **Area using vertices** $(x_1, y_1), (x_2, y_2), (x_3, y_3)$:

$$\text{Area} = \frac{1}{2} \times |x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)|$$

Code Example:

```cpp
#include <iostream>
#include <cmath>

struct Point {
    double x, y;
};

double triangleArea(double a, double b, double c) {
    double s = (a + b + c) / 2.0;
    return std::sqrt(s * (s - a) * (s - b) * (s - c));
}

double triangleAreaFromVertices(const Point& p1, const Point& p2, const Point& p3) {
    return std::abs(p1.x * (p2.y - p3.y) + p2.x * (p3.y - p1.y) + p3.x * (p1.y - p2.y)) /
2.0;
}

int main() {
    // Using side lengths
    double a = 3, b = 4, c = 5;
    std::cout << "Triangle Area (sides): " << triangleArea(a, b, c) << "\n";

    // Using vertices
    Point p1 = {0, 0}, p2 = {3, 0}, p3 = {0, 4};
    std::cout << "Triangle Area (vertices): " << triangleAreaFromVertices(p1, p2, p3) <<
"\n";
}
```

---

2. Rectangle

A rectangle has opposite sides that are equal and parallel.

Properties:

- **Length** l and **Width** w.
- **Area**: $\text{Area} = l \times w$.
- **Perimeter**: $2 \times (l + w)$.

Code Example:

```cpp
#include <iostream>

struct Rectangle {
    double length, width;

    double area() const {
        return length * width;
    }

    double perimeter() const {
        return 2 * (length + width);
    }
};

int main() {
    Rectangle rect = {5, 3};
    std::cout << "Rectangle Area: " << rect.area() << "\n";
    std::cout << "Rectangle Perimeter: " << rect.perimeter() << "\n";
}
```

---

3. Square

A square is a rectangle with all sides equal.

Properties:

- **Side length** s.
- **Area**: $s \times s$.
- **Perimeter**: $4 \times s$.

Code Example:

```cpp
#include <iostream>

struct Square {
    double side;

    double area() const {
        return side * side;
    }

    double perimeter() const {
        return 4 * side;
    }
};

int main() {
    Square sq = {4};
    std::cout << "Square Area: " << sq.area() << "\n";
    std::cout << "Square Perimeter: " << sq.perimeter() << "\n";
}
```

---

4. Circle

A circle is a shape with all points equidistant from the center.

Properties:

- **Radius** r.
- **Area**: $\pi \times r^2$.
- **Circumference**: $2 \times \pi \times r$.

Code Example:

```cpp
#include <iostream>
#define PI 3.14159265358979323846

struct Circle {
    double radius;

    double area() const {
        return PI * radius * radius;
    }

    double circumference() const {
        return 2 * PI * radius;
    }
};

int main() {
    Circle circle = {5};
    std::cout << "Circle Area: " << circle.area() << "\n";
    std::cout << "Circle Circumference: " << circle.circumference() << "\n";
}
```

---

5. Parallelogram

A parallelogram has opposite sides that are parallel and equal in length.

Properties:

- **Base** b, **Height** h.
- **Area**: $b \times h$.
- **Perimeter**: $2 \times (a + b)$, where a and b are adjacent sides.

Code Example:

```cpp
#include <iostream>

struct Parallelogram {
    double base, height, sideA;

    double area() const {
```

```cpp
        return base * height;
    }

    double perimeter() const {
        return 2 * (base + sideA);
    }
};

int main() {
    Parallelogram para = {5, 4, 3};
    std::cout << "Parallelogram Area: " << para.area() << "\n";
    std::cout << "Parallelogram Perimeter: " << para.perimeter() << "\n";
}
```

---

6. Trapezoid

A trapezoid has one pair of opposite sides that are parallel.

Properties:

- **Base1** a, **Base2** b, **Height** h.
- **Area**: $\frac{1}{2} \times (a + b) \times h$.
- **Perimeter**: $a + b + c + d$ (sum of all sides).

Code Example:

```cpp
#include <iostream>

struct Trapezoid {
    double base1, base2, height, sideC, sideD;

    double area() const {
        return 0.5 * (base1 + base2) * height;
    }

    double perimeter() const {
        return base1 + base2 + sideC + sideD;
    }
};

int main() {
    Trapezoid trap = {5, 3, 4, 6, 6};
    std::cout << "Trapezoid Area: " << trap.area() << "\n";
    std::cout << "Trapezoid Perimeter: " << trap.perimeter() << "\n";
}
```

---

7. Regular Polygon (n-sided)

A regular polygon has n equal sides and n equal angles.

Properties:

- **Side length** s.
- **Number of sides** n.

- **Area**: $\frac{n \times s^2}{4 \times \tan(\pi/n)}$.
- **Perimeter**: $n \times s$.

Code Example:

```cpp
#include <iostream>
#include <cmath>

struct RegularPolygon {
    int sides;
    double sideLength;

    double area() const {
        return (sides * sideLength * sideLength) / (4 * std::tan(PI / sides));
    }

    double perimeter() const {
        return sides * sideLength;
    }
};

int main() {
    RegularPolygon pentagon = {5, 4}; // Example for a pentagon
    std::cout << "Polygon Area: " << pentagon.area() << "\n";
    std::cout << "Polygon Perimeter: " << pentagon.perimeter() << "\n";
}
```
