Геометрия — 2

Denis Bakin

План

1. Многоугольники: типы и хранение
2. Площадь треугольника и многоугольника
3. Принадлежность точки треугольнику и выпуклому многоугольнику
4. Проверка на выпуклость
5. Точка в произвольном многоугольнике: метод углов и метод лучей
6. Формула Пика
7. Выпуклая оболочка (опционально)

Что мы умеем (напоминание)

Из прошлой лекции:

- `dot(a, b)` — скалярное произведение, связано с $\cos\theta$
- `cross(a, b)` — векторное произведение, связано с $\sin\theta$
- `length(a)` — длина вектора
- `dist(a, b)` — расстояние между точками

Точка теперь — double

```
struct Point {
    double x = 0.0;
    double y = 0.0;

    Point() = default;
    Point(double x, double y) : x(x), y(y) {}
};
```

- Многие алгоритмы (углы, лучи в случайных направлениях) требуют вещественной арифметики
- Сравнения с нулём — с погрешностью: `std::abs(x) < eps` $(\varepsilon \approx 10^{-9})$

Многоугольник

Многоугольник — часть плоскости, ограниченная замкнутой ломаной

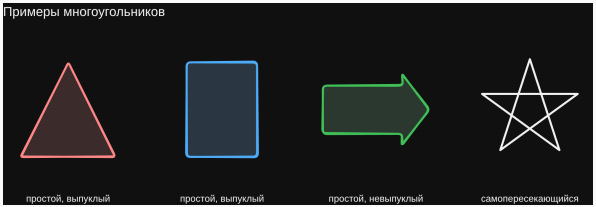

Figure 1: Примеры многоугольников

- *Простой* — ломаная без самопересечений (на картинке все, кроме последнего)
- *Выпуклый* — любой отрезок между двумя точками лежит внутри (первые два)
- *Правильный* — все стороны и углы равны; вписывается в окружность

Многоугольники в программировании

В программировании — последовательность вершин в порядке обхода:

```
std::vector<Point> polygon;
```

Площадь треугольника

Из прошлой лекции: $|\vec{a} \times \vec{b}|$ — площадь параллелограмма

$$S_{\triangle ABC} = \frac{1}{2} \left|(B - A) \times (C - A)\right|$$

Площадь треугольника

Из прошлой лекции: $|\vec{a} \times \vec{b}|$ — площадь параллелограмма

$$S_{\triangle ABC} = \frac{1}{2} \left|(B - A) \times (C - A)\right|$$

Без модуля — **ориентированная** площадь:

- > 0 — обход $A \to B \to C$ против часовой стрелки
- < 0 — по часовой стрелке

```
double triangleArea(Point a, Point b, Point c) {
    return std::abs(cross(b - a, c - a)) / 2.0;
}
```

Площадь многоугольника

Формула Гаусса (shoelace formula):

$$S = \frac{1}{2}\left|\sum_{i=1}^{n} A_i \times A_{i+1}\right|, \quad A_{n+1} = A_1$$

Площадь многоугольника

Формула Гаусса (shoelace formula):

$$S = \frac{1}{2}\left|\sum_{i=1}^{n} A_i \times A_{i+1}\right|, \quad A_{n+1} = A_1$$

Работает для **любого** простого многоугольника, в том числе невыпуклого

Идея вывода

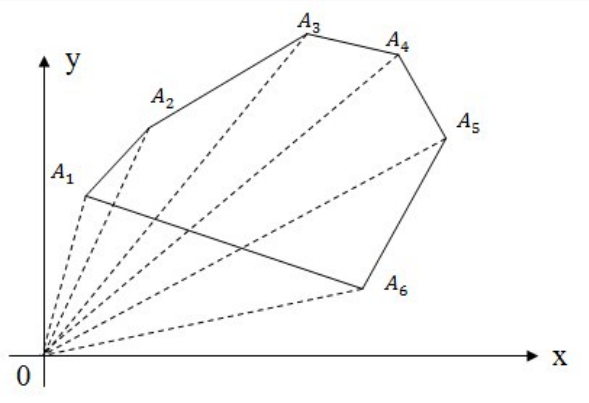

Figure 2: Сумма ориентированных треугольников $\triangle OA_iA_{i+1}$

Лишние участки вне многоугольника учитываются с противоположными знаками — сокращаются

Реализация

```
double polygonArea(const std::vector<Point>& p) {
    int n = (int) p.size();
    double s = 0.0;
    for (int i = 0; i < n; ++i) {
        s += cross(p[i], p[(i + 1) % n]);
    }
    return std::abs(s) / 2.0;
}
```

Реализация

```
double polygonArea(const std::vector<Point>& p) {
    int n = (int) p.size();
    double s = 0.0;
    for (int i = 0; i < n; ++i) {
        s += cross(p[i], p[(i + 1) % n]);
    }
    return std::abs(s) / 2.0;
}
```

`(i + 1) % n` — стандартный приём циклического обхода. Когда `i == n - 1`, индекс возвращается к 0

Альтернатива: метод трапеций

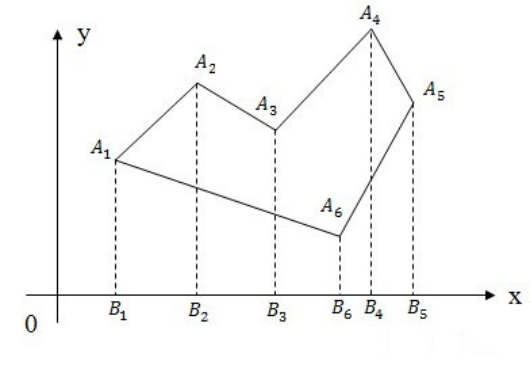

Figure 3: Площадь как сумма трапеций до оси Ox

$$S_{\text{трап}} = \frac{y_i + y_{i+1}}{2} \cdot (x_{i+1} - x_i)$$

«Нижние» трапеции имеют отрицательный знак и сокращают «верхние»

Точка в треугольнике

Идея: P внутри $\triangle ABC$ (обход против ч.с.) $\Leftrightarrow$ P «слева» от каждого ребра

Точка в треугольнике

Идея: P внутри $\triangle ABC$ (обход против ч.с.) $\Leftrightarrow$ P «слева» от каждого ребра

«Слева от ребра» = знак векторного произведения ≥ 0:

$$\begin{cases} (B-A) \times (P-A) \geq 0 \\ (C-B) \times (P-B) \geq 0 \\ (A-C) \times (P-C) \geq 0 \end{cases}$$

Точка в треугольнике

Идея: P внутри $\triangle ABC$ (обход против ч.с.) $\Leftrightarrow$ P «слева» от каждого ребра

«Слева от ребра» = знак векторного произведения ≥ 0:

$$\begin{cases} (B-A) \times (P-A) \geq 0 \\ (C-B) \times (P-B) \geq 0 \\ (A-C) \times (P-C) \geq 0 \end{cases}$$

- Все три строго положительны — точка строго внутри
- Какое-то равно нулю — точка на соответствующем ребре

Точка в треугольнике: код

```
bool pointInTriangle(Point p, Point a, Point b, Point c) {
    const double eps = 1e-9;
    double c1 = cross(b - a, p - a);
    double c2 = cross(c - b, p - b);
    double c3 = cross(a - c, p - c);
    return c1 > -eps && c2 > -eps && c3 > -eps;
}
```

Точка в треугольнике: код

```
bool pointInTriangle(Point p, Point a, Point b, Point c) {
    const double eps = 1e-9;
    double c1 = cross(b - a, p - a);
    double c2 = cross(c - b, p - b);
    double c3 = cross(a - c, p - c);
    return c1 > -eps && c2 > -eps && c3 > -eps;
}
```

Если обход треугольника **по** часовой стрелке — все знаки обратные: `c1 < eps && c2 < eps && c3 < eps`

Точка в выпуклом многоугольнике

Та же идея — проверяем, что точка «слева» от **всех** рёбер обхода

```
bool pointInConvex(Point p, const std::vector<Point>& poly) {
    const double eps = 1e-9;
    int n = (int) poly.size();
    for (int i = 0; i < n; ++i) {
        Point a = poly[i];
        Point b = poly[(i + 1) % n];
        if (cross(b - a, p - a) < -eps) return false;
    }
    return true;
}
```

Время работы — $O(n)$

Проверка на выпуклость

Многоугольник выпуклый $\Leftrightarrow$ при обходе мы всегда «поворачиваем» в одну сторону:

знак $(b - a) \times (c - b)$ не меняется по всем тройкам соседних вершин

Проверка на выпуклость

Многоугольник выпуклый $\Leftrightarrow$ при обходе мы всегда «поворачиваем» в одну сторону:

знак $(b-a)\times(c-b)$ не меняется по всем тройкам соседних вершин

```
bool isConvex(const std::vector<Point>& poly) {
    int n = (int) poly.size(), sign = 0;
    for (int i = 0; i < n; ++i) {
        Point a = poly[i], b = poly[(i + 1) % n], c = poly[(i + 2) % n];
        double v = cross(b - a, c - b);
        if (v > 0) { if (sign < 0) return false; sign = 1; }
        else if (v < 0) { if (sign > 0) return false; sign = -1; }
    }
    return true;
}
```

Точка в произвольном многоугольнике

Для невыпуклых «слева от всех» **не работает**

Точка в произвольном многоугольнике

Для невыпуклых «слева от всех» **не работает**

Два стандартных подхода, оба за $O(n)$ на запрос:

1. **Метод суммы углов** — ориентированные углы, под которыми видны рёбра
2. **Метод лучей** — сколько раз луч из точки пересекает границу

Метод суммы углов

Для каждого ребра A_iA_{i+1} — ориентированный угол, под которым оно видно из P:

$$\theta = \sum_{i=1}^{n} \measuredangle(A_i - P,\ A_{i+1} - P)$$

Метод суммы углов

Для каждого ребра A_iA_{i+1} — ориентированный угол, под которым оно видно из P:

$$\theta = \sum_{i=1}^{n} \angle(A_i - P,\ A_{i+1} - P)$$

- $|\theta| \approx 2\pi$ — точка **внутри** многоугольника
- $\theta \approx 0$ — точка **снаружи**

Метод суммы углов: почему

Внутри: луч $P \to A_i$ совершает один полный оборот при обходе $\Rightarrow$ сумма углов $= \pm 2\pi$ (знак — направление обхода)

Метод суммы углов: почему

Внутри: луч $P \to A_i$ совершает один полный оборот при обходе $\Rightarrow$ сумма углов $= \pm 2\pi$ (знак — направление обхода)

Снаружи: направление «отклоняется» в одну сторону и возвращается обратно $\Rightarrow$ сумма ≈ 0

Метод суммы углов: код

Ориентированный угол — из прошлой лекции:

```
double angleBetween(Point a, Point b) {
    return std::atan2(cross(a, b), dot(a, b));
}
```

Метод суммы углов: код

Ориентированный угол — из прошлой лекции:

```
double angleBetween(Point a, Point b) {
    return std::atan2(cross(a, b), dot(a, b));
}
```

```
bool pointInPolygonAngle(Point p, const std::vector<Point>& poly) {
    int n = (int) poly.size();
    double total = 0.0;
    for (int i = 0; i < n; ++i) {
        total += angleBetween(poly[i] - p, poly[(i + 1) % n] - p);
    }
    return std::abs(total) > M_PI;
}
```

Порог π — любое значение строго между 0 и 2π

Метод лучей

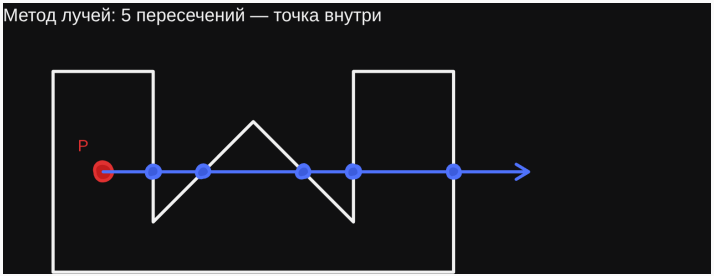

Figure 4: Луч из точки P пересекает границу многоугольника

Метод лучей

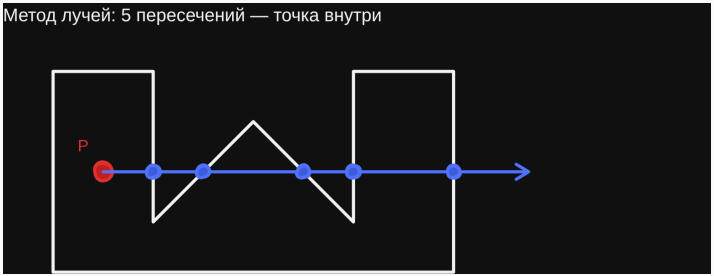

Figure 4: Луч из точки P пересекает границу многоугольника

- Точка **внутри** $\Rightarrow$ нечётное число пересечений
- Точка **снаружи** $\Rightarrow$ чётное

Метод лучей: реализация

```
bool pointInPolygonRay(Point p, const std::vector<Point>& poly) {
    // "случайное" направление, в простом случае мы сами его выбираем
    Point d = {1.7, 2.3};
    size_t n = poly.size();
    size_t crossings = 0;
    for (size_t i = 0; i < n; ++i) {
        if (raySegmentIntersect(p, d, poly[i], poly[(i + 1) % n])) {
            ++crossings;
        }
    }
    return crossings % 2 == 1;
}
```

Метод лучей: реализация

```
bool pointInPolygonRay(Point p, const std::vector<Point>& poly) {
    // "случайное" направление, в простом случае мы сами его выбираем
    Point d = {1.7, 2.3};
    size_t n = poly.size();
    size_t crossings = 0;
    for (size_t i = 0; i < n; ++i) {
        if (raySegmentIntersect(p, d, poly[i], poly[(i + 1) % n])) {
            ++crossings;
        }
    }
    return crossings % 2 == 1;
}
```

«Случайное» направление — чтобы луч случайно не прошёл через вершину или вдоль ребра

Пересечение луча и отрезка

Пересечение луча и отрезка

$$P + t \cdot \vec{d} = A + s \cdot (B - A), \quad t \geq 0,\ s \in [0, 1]$$

- Это система из двух уравнений (по одному на каждую координату) с двумя неизвестными t и s
- Решается так же, как пересечение прямых из прошлой лекции
- В конце — проверка $t \geq -\varepsilon$ и $-\varepsilon \leq s \leq 1 + \varepsilon$

Формула Пика

Для многоугольника, **все** вершины которого имеют целые координаты:

$$S = V + \frac{E}{2} - 1$$

- V — число целых точек **внутри** многоугольника
- E — число целых точек **на границе** (включая вершины)

Формула Пика: пример

Квадрат с вершинами $(0,0)$, $(3,0)$, $(3,3)$, $(0,3)$:

- Площадь: $S = 9$
- Внутренние точки: $(1,1), (1,2), (2,1), (2,2) \Rightarrow V = 4$
- Точки на границе: по 4 на каждой стороне $\Rightarrow E = 12$

Формула Пика: пример

Квадрат с вершинами $(0,0)$, $(3,0)$, $(3,3)$, $(0,3)$:

- Площадь: $S = 9$
- Внутренние точки: $(1,1), (1,2), (2,1), (2,2) \Rightarrow V = 4$
- Точки на границе: по 4 на каждой стороне $\Rightarrow E = 12$

$$V + \frac{E}{2} - 1 = 4 + 6 - 1 = 9 = S \quad \checkmark$$

Формула Пика: пример

Квадрат с вершинами $(0,0)$, $(3,0)$, $(3,3)$, $(0,3)$:

- Площадь: $S = 9$
- Внутренние точки: $(1,1), (1,2), (2,1), (2,2) \Rightarrow V = 4$
- Точки на границе: по 4 на каждой стороне $\Rightarrow E = 12$

$$V + \frac{E}{2} - 1 = 4 + 6 - 1 = 9 = S \quad \checkmark$$

Применение: если знаем S и E, находим V (E на отрезке (x_1, y_1)—(x_2, y_2) равно $\gcd(|\Delta x|, |\Delta y|) + 1$)

Выпуклая оболочка

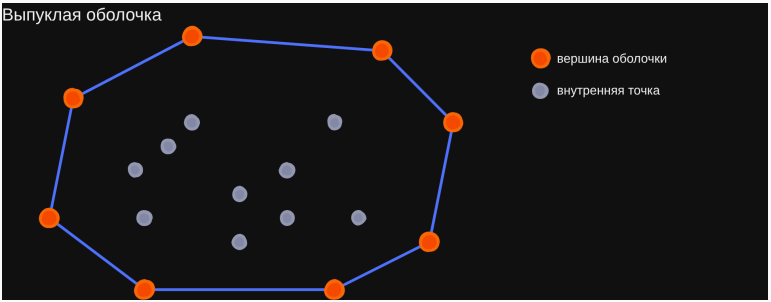

Figure 5: Резинка, натянутая на множество гвоздей

Выпуклая оболочка — наименьший выпуклый многоугольник, содержащий все заданные точки

Можно найти за $O(n \log n)$ или $O(nh)$, где h — число вершин оболочки

Применения:

- Проверка точки в облаке точек — через принадлежность оболочке за $O(\log n)$
- Диаметр множества (самая дальняя пара точек)

Итоги

- **Площадь** многоугольника: формула Гаусса — сумма $A_i \times A_{i+1}$
- **Точка в треугольнике / выпуклом**: знак cross по всем рёбрам
- **Точка в произвольном** многоугольнике: сумма углов или метод лучей
- **Выпуклость**: знак cross не меняется по всем тройкам вершин
- **Формула Пика**: $S = V + \frac{E}{2} - 1$ — связь площади и целых точек
- **Выпуклая оболочка** — фундамент вычислительной геометрии