Устойчивая реализация алгоритмов вычислительной геометрии

Антон Ковалев

ЗАО Транзас, СПбГУ ИТМО

октябрь 2011

Расчеты в вычислительной геометрии

Реализация алгоритмов вычислительной геометрии зачастую очень сложна, так как:

Расчеты в вычислительной геометрии

Реализация алгоритмов вычислительной геометрии зачастую очень сложна, так как:

- сложен сам алгоритм (например, incremental segment Voronoi diagram);

Расчеты в вычислительной геометрии

Реализация алгоритмов вычислительной геометрии зачастую очень сложна, так как:

- сложен сам алгоритм (например, incremental segment Voronoi diagram);
- в описании алгоритма опущены вырожденные случаи (например, Bentley-Ottmann);

Расчеты в вычислительной геометрии

Реализация алгоритмов вычислительной геометрии зачастую очень сложна, так как:

- сложен сам алгоритм (например, incremental segment Voronoi diagram);
- в описании алгоритма опущены вырожденные случаи (например, Bentley-Ottmann);
- вычисления с плавающей точкой дают погрешность, что приводит к нарушению инвариантов алгоритма.

Расчеты в вычислительной геометрии

Реализация алгоритмов вычислительной геометрии зачастую очень сложна, так как:

- сложен сам алгоритм (например, incremental segment Voronoi diagram);
- в описании алгоритма опущены вырожденные случаи (например, Bentley-Ottmann);
- вычисления с плавающей точкой дают погрешность, что приводит к нарушению инвариантов алгоритма.

Рассмотрим один из способов справиться с последней проблемой: фильтрованное вычисление предикатов.

Определение пересечения двух отрезков на плоскости

Определение

Отрезок s задается начальной и конечной точками $a, b \in \mathbb{R}^2$ и определяется как множество точек $s = \{(1-t)a + tb, t \in [0;1]\}$.

Определение пересечения двух отрезков на плоскости

Определение

Отрезок s задается начальной и конечной точками $a, b \in \mathbb{R}^2$ и определяется как множество точек $s = \{(1-t)a + tb, t \in [0;1]\}$.

Определение

Отрезки s_1 и s_2 пересекаются, если существует непустое множество общих точек.

Определение пересечения двух отрезков на плоскости

Определение

$$Turn(s, c) = \begin{cases} 1 & \text{точка } c \text{ лежит слева от направленного отрезка } s, \\ -1 & \text{точка } c \text{ лежит справа от направленного отрезка } s, \\ 0 & s \text{ и } c \text{ коллинеарны.} \end{cases}$$

Определение пересечения двух отрезков на плоскости

Определение

$$Turn(s, c) = \begin{cases} 1 & \text{точка } c \text{ лежит слева от направленного отрезка } s, \\ -1 & \text{точка } c \text{ лежит справа от направленного отрезка } s, \\ 0 & s \text{ и } c \text{ коллинеарны.} \end{cases}$$

Утверждение

$$s_1 \bigcap s_2 \iff \begin{cases} Turn(s_1, s_{2,a}) \cdot Turn(s_1, s_{2,b}) \leq 0 \\ Turn(s_2, s_{1,a}) \cdot Turn(s_2, s_{1,b}) \leq 0\,, \\ AABB(s_1) \bigcap AABB(s_2) \neq \varnothing \end{cases}$$

где $AABB$ — это axis-aligned bounding box.

Вычисление предиката *Turn*

Утверждение

$$Turn(s, c) = sign\big((s_b - s_a) \times (c - s_a)\big) = sign\left(\begin{vmatrix} s_{b,x} - s_{a,x} & c_x - s_{a,x} \\ s_{b,y} - s_{a,y} & c_y - s_{a,y} \end{vmatrix}\right)$$

Вычисление предиката *Turn*

Утверждение

$$Turn(s, c) = sign\big((s_b - s_a) \times (c - s_a)\big) = sign\left(\begin{vmatrix} s_{b,x} - s_{a,x} & c_x - s_{a,x} \\ s_{b,y} - s_{a,y} & c_y - s_{a,y} \end{vmatrix}\right)$$

Проблема?

Вычисление предиката *Turn*

Утверждение

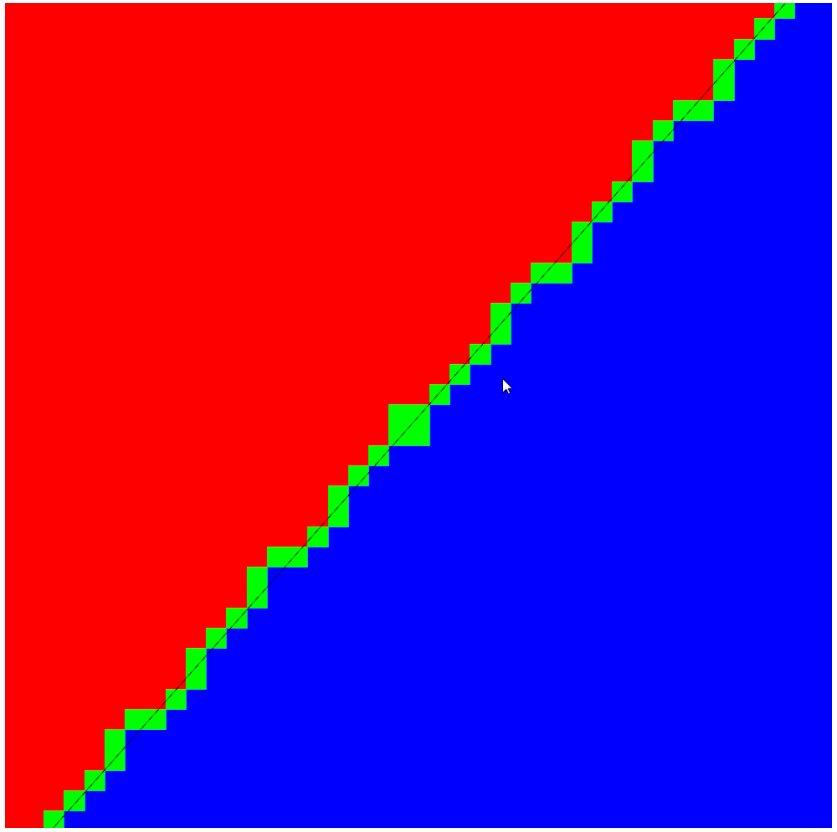
$$Turn(s, c) = sign\big((s_b - s_a) \times (c - s_a)\big) = sign\left(\begin{vmatrix} s_{b,x} - s_{a,x} & c_x - s_{a,x} \\ s_{b,y} - s_{a,y} & c_y - s_{a,y} \end{vmatrix}\right)$$

Проблема? Вычисления с плавающей точкой:

```
point_2 a(3.0, 5.0);
point_2 b = -a;
point_2 c =  a * (1LL << 52);

// Что вернет эта функция?
Turn(segment_2(a, b), c);
```

Вычисление предиката *Turn*



Вычисление поворота от отрезка и точки плоскости: красный цвет — левый поворот, синий — правый, зеленый — точка и отрезок принадлежат одной прямой.

Вычисление предиката *Turn*

Числа с плавающей точкой

Двоичное число с плавающей точкой:
$a = \pm 1.a_1 a_2 a_3 \ldots a_{p-1} \times 2^{\gamma}, a_i \in \{0, 1\}$.

Вычисление предиката *Turn*

Числа с плавающей точкой

Двоичное число с плавающей точкой:
$a = \pm 1.a_1 a_2 a_3 \ldots a_{p-1} \times 2^{\gamma}, a_i \in \{0, 1\}$.

Определение

$ulp(a) = 2^{-p+1+\gamma}$

Вычисление предиката *Turn*

Числа с плавающей точкой

Двоичное число с плавающей точкой:
$a = \pm 1.a_1 a_2 a_3 \ldots a_{p-1} \times 2^{\gamma}, a_i \in \{0, 1\}$.

Определение

$ulp(a) = 2^{-p+1+\gamma}$

D — множество всех чисел с плавающей точкой с операциями $\oplus, \ominus, \otimes$.

Вычисление предиката *Turn*

Числа с плавающей точкой

Двоичное число с плавающей точкой:
$a = \pm 1.a_1 a_2 a_3 \ldots a_{p-1} \times 2^{\gamma}, a_i \in \{0, 1\}$.

Определение

$ulp(a) = 2^{-p+1+\gamma}$

D — множество всех чисел с плавающей точкой с операциями $\oplus, \ominus, \otimes$.
$\forall a, b \in D,\ \varepsilon = 2^{-p}$:

- $a \oplus b = (a + b)(1 + \delta),\ |\delta| \leq \varepsilon$
- $a \ominus b = (a - b)(1 + \delta),\ |\delta| \leq \varepsilon$
- $a \otimes b = ab(1 + \delta),\ |\delta| \leq \varepsilon$

Вычисление предиката *Turn*

Числа с плавающей точкой

Двоичное число с плавающей точкой:
$a = \pm 1.a_1 a_2 a_3 \ldots a_{p-1} \times 2^{\gamma}, a_i \in \{0, 1\}$.

Определение

$ulp(a) = 2^{-p+1+\gamma}$

D — множество всех чисел с плавающей точкой с операциями $\oplus, \ominus, \otimes$.
$\forall a, b \in D,\ \varepsilon = 2^{-p}$:

- $a \oplus b = (a + b)(1 + \delta),\ |\delta| \leq \varepsilon$
- $a \ominus b = (a - b)(1 + \delta),\ |\delta| \leq \varepsilon$
- $a \otimes b = ab(1 + \delta),\ |\delta| \leq \varepsilon$

Утверждение

$\forall a, b, c \in D^2,\ \tilde{v} = (b_x \ominus a_x) \otimes (c_y \ominus a_y) \ominus (b_y \ominus a_y) \otimes (c_x \ominus a_x)$

$$\exists \tilde{\epsilon} \in D : \begin{cases} \tilde{v} > \tilde{\epsilon} \Rightarrow (b - a) \times (c - a) > 0 \\ \tilde{v} < -\tilde{\epsilon} \Rightarrow (b - a) \times (c - a) < 0 \end{cases}$$

Вычисление предиката *Turn*

Расчет $\tilde{\epsilon}$

$$\overbrace{(b-a)\times(c-a)}^{v} \approx \overbrace{(b_x \ominus a_x)\otimes(c_y \ominus a_y)\ominus(b_y \ominus a_y)\otimes(c_x \ominus a_x)}^{\tilde{v}}$$
$$= \big[(b_x - a_x)(c_y - a_y)(1+\delta_1)(1+\delta_2)(1+\delta_3)$$
$$-(b_y - a_y)(c_x - a_x)(1+\delta_4)(1+\delta_5)(1+\delta_6)\big](1+\delta_7),$$

Вычисление предиката *Turn*

Расчет $\tilde{\epsilon}$

$$\overbrace{(b-a)\times(c-a)}^{v} \approx \overbrace{(b_x \ominus a_x)\otimes(c_y \ominus a_y)\ominus(b_y \ominus a_y)\otimes(c_x \ominus a_x)}^{\tilde{v}}$$
$$= \big[(b_x-a_x)(c_y-a_y)(1+\delta_1)(1+\delta_2)(1+\delta_3)$$
$$-(b_y-a_y)(c_x-a_x)(1+\delta_4)(1+\delta_5)(1+\delta_6)\big](1+\delta_7),$$

$|\delta_i| \le \varepsilon$;

Вычисление предиката *Turn*

Расчет $\tilde{\epsilon}$

$$\overbrace{(b-a)\times(c-a)}^{v} \approx \overbrace{(b_x \ominus a_x)\otimes(c_y \ominus a_y)\ominus(b_y \ominus a_y)\otimes(c_x \ominus a_x)}^{\tilde{v}}$$
$$= \big[(b_x-a_x)(c_y-a_y)(1+\delta_1)(1+\delta_2)(1+\delta_3)$$
$$-(b_y-a_y)(c_x-a_x)(1+\delta_4)(1+\delta_5)(1+\delta_6)\big](1+\delta_7),$$

$|\delta_i| \le \varepsilon$;

$$\epsilon = |v-\tilde{v}| \le \overbrace{\big(|(b_x-a_x)(c_y-a_y)| + |(b_y-a_y)(c_x-a_x)|\big)}^{e} \times$$
$$\times\,(4\varepsilon + 6\varepsilon^2 + 4\varepsilon^3 + \varepsilon^4)$$

Вычисление предиката *Turn*

Расчет $\tilde{\epsilon}$

$$\overbrace{(b-a)\times(c-a)}^{v} \approx \overbrace{(b_x \ominus a_x)\otimes(c_y \ominus a_y)\ominus(b_y \ominus a_y)\otimes(c_x \ominus a_x)}^{\tilde{v}}$$
$$= \big[(b_x-a_x)(c_y-a_y)(1+\delta_1)(1+\delta_2)(1+\delta_3)$$
$$-(b_y-a_y)(c_x-a_x)(1+\delta_4)(1+\delta_5)(1+\delta_6)\big](1+\delta_7),$$

$|\delta_i| \le \varepsilon$;

$$\epsilon = |v-\tilde{v}| \le \overbrace{\big(|(b_x-a_x)(c_y-a_y)| + |(b_y-a_y)(c_x-a_x)|\big)}^{e} \times$$
$$\times\,(4\varepsilon + 6\varepsilon^2 + 4\varepsilon^3 + \varepsilon^4)$$

Вычислим выражение справа в модели floating-point.

Вычисление предиката *Turn*

Расчет $\tilde{\epsilon}$

$$\overbrace{\big(|(b_x - a_x)(c_y - a_y)| + |(b_y - a_y)(c_x - a_x)|\big)}^{e}(1-\varepsilon)^4 \leq$$

$$\leq \overbrace{|(b_x \ominus a_x) \otimes (c_y \otimes a_y)| \oplus |(b_y \ominus a_y) \otimes (c_x \ominus a_x)|}^{\tilde{e}}$$

Вычисление предиката *Turn*

Расчет $\tilde{\epsilon}$

$$\overbrace{\big(|(b_x - a_x)(c_y - a_y)| + |(b_y - a_y)(c_x - a_x)|\big)}^{e}(1-\varepsilon)^4 \le$$

$$\le \overbrace{|(b_x \ominus a_x) \otimes (c_y \otimes a_y)| \oplus |(b_y \ominus a_y) \otimes (c_x \ominus a_x)|}^{\tilde{e}}$$

$$e \le \tilde{e}\frac{1}{(1-\varepsilon)^4} = \tilde{e}(1 + 4\varepsilon + 10\varepsilon^2 + 20\varepsilon^3 + \cdots)$$

Вычисление предиката *Turn*

Расчет $\tilde{\epsilon}$

$$\overbrace{\big(|(b_x - a_x)(c_y - a_y)| + |(b_y - a_y)(c_x - a_x)|\big)}^{e}(1-\varepsilon)^4 \;\leq$$

$$\leq \overbrace{|(b_x \ominus a_x) \otimes (c_y \otimes a_y)| \oplus |(b_y \ominus a_y) \otimes (c_x \ominus a_x)|}^{\tilde{e}}$$

$$e \leq \tilde{e}\frac{1}{(1-\varepsilon)^4} = \tilde{e}(1 + 4\varepsilon + 10\varepsilon^2 + 20\varepsilon^3 + \cdots)$$

$$\epsilon \;\leq\; \tilde{\epsilon} \;\leq\; \tilde{e}(1 + 4\varepsilon + 10\varepsilon^2 + 20\varepsilon^3 + \cdots)(4\varepsilon + 6\varepsilon^2 + 4\varepsilon^3 + \varepsilon^4)$$

Вычисление предиката *Turn*

Расчет $\tilde{\epsilon}$

$$\overbrace{\big(|(b_x - a_x)(c_y - a_y)| + |(b_y - a_y)(c_x - a_x)|\big)}^{e}(1-\varepsilon)^4 \;\leq$$

$$\leq \overbrace{|(b_x \ominus a_x) \otimes (c_y \otimes a_y)| \oplus |(b_y \ominus a_y) \otimes (c_x \ominus a_x)|}^{\tilde{e}}$$

$$e \leq \tilde{e}\frac{1}{(1-\varepsilon)^4} = \tilde{e}(1 + 4\varepsilon + 10\varepsilon^2 + 20\varepsilon^3 + \cdots)$$

$$\epsilon \;\leq\; \tilde{\epsilon} \;\leq\; \tilde{e}(1 + 4\varepsilon + 10\varepsilon^2 + 20\varepsilon^3 + \cdots)(4\varepsilon + 6\varepsilon^2 + 4\varepsilon^3 + \varepsilon^4)$$

$$\boxed{\tilde{\epsilon} < 8\varepsilon\tilde{e}}$$

Вычисление предиката *Turn*

Утверждение

$$\begin{cases} \tilde{v} > 8\varepsilon\tilde{e} \Rightarrow v > 0 \\ \tilde{v} < -8\varepsilon\tilde{e} \Rightarrow v < 0 \end{cases}$$

Вычисление предиката *Turn*

Утверждение

$$\begin{cases} \tilde{v} > 8\varepsilon\tilde{e} \Rightarrow v > 0 \\ \tilde{v} < -8\varepsilon\tilde{e} \Rightarrow v < 0 \end{cases}$$

Что делать, если $\tilde{v} \in [-8\varepsilon\tilde{e}; 8\varepsilon\tilde{e}]$?

Вычисление предиката *Turn*

Интервальная арифметика

Определение

Интервальная арифметика:

$$a \in [a_{lo}, a_{hi}];$$

$$a \cdot b \in \left[\min_{\substack{a' \in [a_{lo}; a_{hi}] \\ b' \in [b_{lo}; b_{hi}]}} \left\{ a' \underline{\odot} b' \right\}; \max_{\substack{a' \in [a_{lo}; a_{hi}] \\ b' \in [b_{lo}; b_{hi}]}} \left\{ a' \overline{\odot} b' \right\} \right],$$

$$\cdot \in [+, -, \times],$$

$\odot \in [\oplus, \ominus, \otimes]$ с округлением, соответственно, к $-\infty$ и $+\infty$.

Замечание

$sqr([a_{lo}, a_{hi}]) = ?$

Вычисление предиката *Turn*
Интервальная арифметика

Определение

Интервальная арифметика:

$$a \in [a_{lo}, a_{hi}];$$

$$a \cdot b \in \left[\min_{\substack{a' \in [a_{lo}; a_{hi}] \\ b' \in [b_{lo}; b_{hi}]}} \left\{ a' \underline{\odot} b' \right\}; \max_{\substack{a' \in [a_{lo}; a_{hi}] \\ b' \in [b_{lo}; b_{hi}]}} \left\{ a' \overline{\odot} b' \right\} \right],$$

$$\cdot \in [+, -, \times],$$

$\odot \in [\oplus, \ominus, \otimes]$ с округлением, соответственно, к $-\infty$ и $+\infty$.

Замечание

$sqr([a_{lo}, a_{hi}]) = ?$

Вычисление предиката *Turn*

Интервальная арифметика

Утверждение

Если интервал $v = (b - a) \times (c - a)$ не содержит 0, знак v вычислен точно.

Вычисление предиката *Turn*

Интервальная арифметика

Утверждение

Если интервал $v = (b - a) \times (c - a)$ не содержит 0, знак v вычислен точно.

Замечание

Переключать режим округления процессора дорого.

Вычисление предиката *Turn*

Интервальная арифметика

Утверждение

Если интервал $v = (b - a) \times (c - a)$ не содержит 0, знак v вычислен точно.

Замечание

Переключать режим округления процессора дорого.
$[a_{lo}; a_{hi}] = -[-a_{hi}; -a_{lo}]$.

Вычисление предиката *Turn*

Интервальная арифметика

Утверждение

Если интервал $v = (b - a) \times (c - a)$ не содержит 0, знак v вычислен точно.

Замечание

Переключать режим округления процессора дорого.
$[a_{lo}; a_{hi}] = -[-a_{hi}; -a_{lo}]$.

Эффективная реализация есть, например, в boost/interval.

Вычисление предиката *Turn*
Интервальная арифметика

Утверждение

Если интервал $v = (b - a) \times (c - a)$ не содержит 0, знак v вычислен точно.

Замечание

Переключать режим округления процессора дорого.
$[a_{lo}; a_{hi}] = -[-a_{hi}; -a_{lo}]$.

Эффективная реализация есть, например, в boost/interval.
Что делать, если интервал v содержит 0?

Вычисление предиката *Turn*
Рациональная арифметика

Определение

Рациональная арифметика: число представляется в виде рациональной дроби с длинным числителем и длинным знаменателем.

Вычисление предиката *Turn*

Рациональная арифметика

Определение

Рациональная арифметика: число представляется в виде рациональной дроби с длинным числителем и длинным знаменателем.

Замечание

Недостатки:

- даже эффективные реализации сложны и медленны;
- требует или сложной разработки, или добавления еще одной зависимости (gmp, например).

Вычисление предиката *Turn*

Adaptive precision floating-point arithmetic

Идея

$\forall a, b \in D \; \exists c \in D :$

- $a + b = a \oplus b + c, |c| \leq \frac{1}{2} ulp\,(a \oplus b)$

Вычисление предиката *Turn*

Adaptive precision floating-point arithmetic

Идея

$\forall a, b \in D\ \exists c \in D:$

- $a + b = a \oplus b + c, |c| \leq \frac{1}{2} ulp\,(a \oplus b)$
- $ab = a \otimes b + c, |c| \leq \frac{1}{2} ulp\,(a \otimes b)$

Вычисление предиката *Turn*

Adaptive precision floating-point arithmetic

Идея

$\forall a, b \in D\ \exists c \in D:$

- $a + b = a \oplus b + c, |c| \leq \frac{1}{2} ulp\,(a \oplus b)$
- $ab = a \otimes b + c, |c| \leq \frac{1}{2} ulp\,(a \otimes b)$

$$\forall b = f(a_1, a_2, \ldots, a_n)_{a_i \in D}\ \exists \{c_j\} : \begin{cases} c_j \in D \\ |c_j| \leq \frac{1}{2} ulp(c_{j+1}), \\ b = \sum c_j \end{cases}$$

где f — это арифметическое выражение с операциями $+$, $-$ и $\times$.

Вычисление предиката *Turn*

Adaptive precision floating-point arithmetic

```
double sum(double a, double b
           , double & roundoff)
{
    double res = a + b;
    double bv  = res - a;
    double av  = res - bv;
    double br  = b - bv;
    double ar  = a - av;
    roundoff   = ar + br;
    return res;
}
```

Вычисление предиката *Turn*

Adaptive precision floating-point arithmetic

```
double mul(double a, double b, double & roundoff)
{
    double res = a * b;
    size_t s = std::numeric_limits<double>::digits / 2
             + std::numeric_limits<double>::digits % 2;

    double a_hi, a_lo, b_hi, b_lo;
    impl::split(a, s, a_hi, a_lo);
    impl::split(b, s, b_hi, b_lo);

    double e1 = res - (a_hi * b_hi);
    double e2 = e1 - (a_lo * b_hi);
    double e3 = e2 - (b_lo * a_hi);

    roundoff = (a_lo * b_lo) - e3;
    return res;
}
```

Вычисление предиката *Turn*

Adaptive precision floating-point arithmetic

```
void split(double a, size_t s
           , double & hi, double & lo)
{
    double c  = ((1LL << s) + 1LL) * a;
    double ab = c - a;
    hi = c - ab;
    lo = a - hi;
}
```

Вычисление предиката *Turn*

Adaptive precision floating-point arithmetic

```
template <size_t N> struct grow_expansion_f
{
    static void calc(double const * e, double b
                     , double * r)
    {
        b = sum(*e, b, *r);
        grow_expansion_f<N - 1>::calc(e + 1, b, r + 1);
    }
};

template <> struct grow_expansion_f<0>
{
    static void calc(double const * e, double b
                     , double * r)
    { *r = b; }
};
```

Вычисление предиката *Turn*

Adaptive precision floating-point arithmetic

```
template <size_t N1, size_t N2> struct expand_sum_f
{
    static void calc(double const * e, double const * f
                     , double * r)
    {
        grow_expansion<N1>(e, *f, r);
        expand_sum_f<N1, N2 - 1>::calc(r + 1
                                       , f + 1, r + 1);
    }
};

template <size_t N1> struct expand_sum_f<N1, 0>
{
    static void calc(double const * e, double const * f
                     , double * r) {}
};
```

Вычисление предиката *Turn*

Adaptive precision floating-point arithmetic

```
double sa[12];
sa[1]  = mul( b.x, c.y, sa[0]);
...
sa[11] = mul( b.y, a.x, sa[10]);

double sb[12];
expand_sum<2, 2>(sa + 0, sa + 2,  sb);
expand_sum<2, 2>(sa + 4, sa + 6,  sb + 4);
expand_sum<2, 2>(sa + 8, sa + 10, sb + 8);

double sc[8];
expand_sum<4, 4>(sb, sb + 4, sc);

double sd[12];
expand_sum<8, 4>(sc, sb + 8, sd);

return sign<12>(sd);
```

Фильтрованное вычисление предиката

Эффективное и точное решение:

- числа с плавающей точкой;

Фильтрованное вычисление предиката

Эффективное и точное решение:

- числа с плавающей точкой;
- интервальная арифметика;

Фильтрованное вычисление предиката

Эффективное и точное решение:

- числа с плавающей точкой;
- интервальная арифметика;
- рациональная или адаптивная арифметика, ESSA, etc.

Фильтрованное вычисление предиката

Эффективное и точное решение:

- числа с плавающей точкой;
- интервальная арифметика;
- рациональная или адаптивная арифметика, ESSA, etc.

Замечание

Большинство предикатов в вычислительной геометрии сводятся к *sign* от элементарного арифметического выражения.

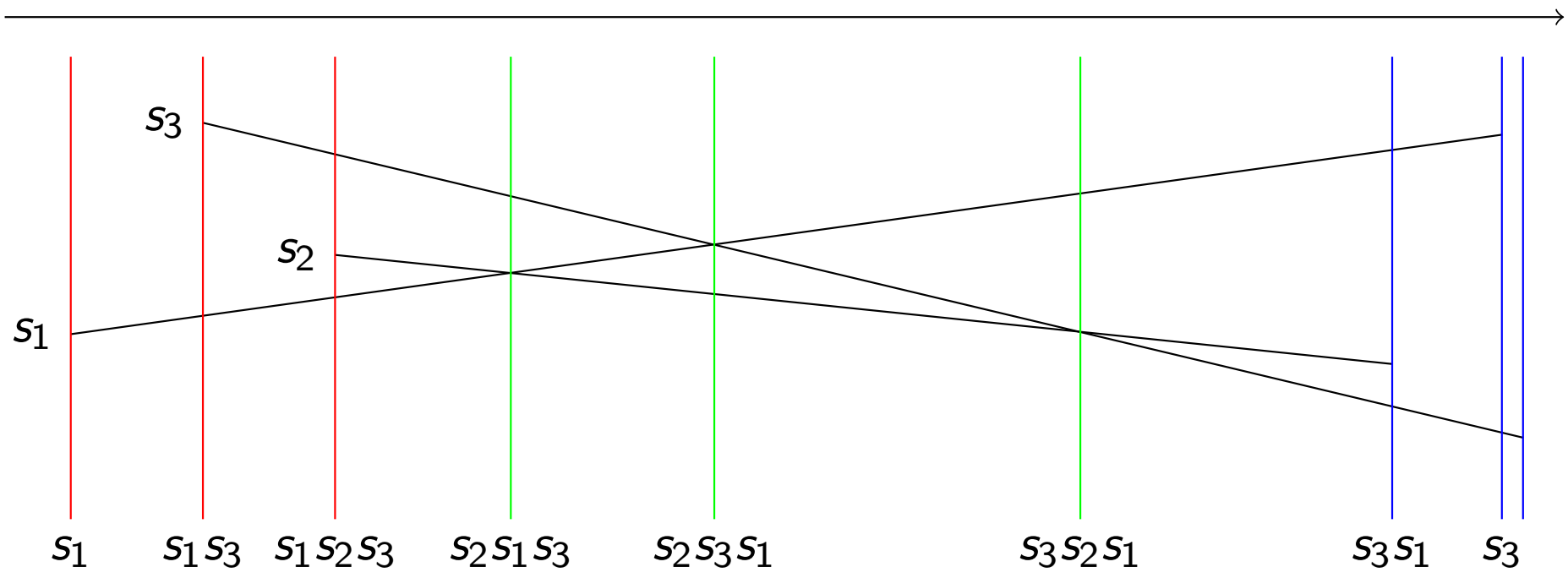
Алгоритм Бентли-Оттмана

$S = \{s\}$ – множество отрезков, $n = \|S\|$. Требуется найти множество $R = \{r_i\}$, где $r_i = \{s_{i,k}\}$ множество всех отрезков, проходящих через некоторую точку q_i.

Алгоритм Бентли-Оттмана

$S = \{s\}$ – множество отрезков, $n = \|S\|$. Требуется найти множество $R = \{r_i\}$, где $r_i = \{s_{i,k}\}$ множество всех отрезков, проходящих через некоторую точку q_i.

Идея: заметающая прямая, $O((n+k)\log n)$ (k – число пересечений).

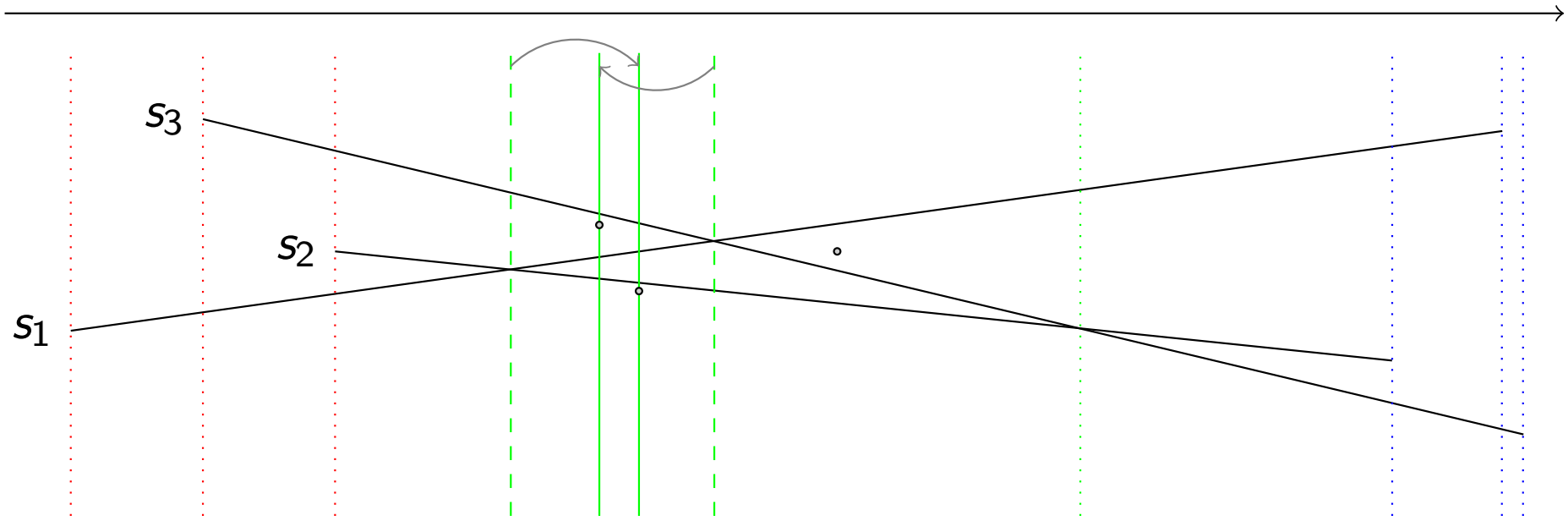


Алгоритм Бентли-Оттмана

Все хорошо?

Алгоритм Бентли-Оттмана

Все хорошо? Вот, что будет, если добавлять в очередь событий координаты точек пересечений отрезков:



Алгоритм Бентли-Оттмана

А нужны ли сами точки пересечений?

Алгоритм Бентли-Оттмана

А нужны ли сами точки пересечений? Необходимо поддерживать корректный порядок событий:

$$e_1 < e_2 \Leftrightarrow \left[\begin{array}{r} e_{1,x} < e_{2,x} \\ e_{1,x} = e_{2,x} \ \wedge \ e_{1,y} < e_{2,y} \end{array} \right.$$

Алгоритм Бентли-Оттмана

А нужны ли сами точки пересечений? Необходимо поддерживать корректный порядок событий:

$$e_1 < e_2 \Leftrightarrow \left[\begin{array}{l} e_{1,x} < e_{2,x} \\ e_{1,x} = e_{2,x} \ \wedge \ e_{1,y} < e_{2,y} \end{array} \right.$$

Найдем точку пересечения q двух непараллельных отрезков $a_1 b_1$ и $a_2 b_2$:

$$\begin{pmatrix} a_{1,y} - b_{1,y} & b_{1,x} - a_{1,x} \\ a_{2,y} - b_{2,y} & b_{2,x} - a_{2,x} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} (a_{1,y} - b_{1,y}) a_{1,x} + (b_{1,x} - a_{1,x}) a_{1,y} \\ (a_{2,y} - b_{2,y}) a_{2,x} + (b_{2,x} - a_{2,x}) a_{2,y} \end{pmatrix}$$

Алгоритм Бентли-Оттмана

А нужны ли сами точки пересечений? Необходимо поддерживать корректный порядок событий:

$$e_1 < e_2 \Leftrightarrow \left[\begin{array}{l} e_{1,x} < e_{2,x} \\ e_{1,x} = e_{2,x} \ \wedge \ e_{1,y} < e_{2,y} \end{array}\right.$$

Найдем точку пересечения q двух непараллельных отрезков $a_1 b_1$ и $a_2 b_2$:

$$\begin{pmatrix} a_{1,y} - b_{1,y} & b_{1,x} - a_{1,x} \\ a_{2,y} - b_{2,y} & b_{2,x} - a_{2,x} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} (a_{1,y} - b_{1,y})a_{1,x} + (b_{1,x} - a_{1,x})a_{1,y} \\ (a_{2,y} - b_{2,y})a_{2,x} + (b_{2,x} - a_{2,x})a_{2,y} \end{pmatrix}$$

$$\begin{pmatrix} x \\ y \end{pmatrix} = \frac{\begin{pmatrix} b_{2,x} - a_{2,x} & a_{1,x} - b_{1,x} \\ b_{2,y} - a_{2,y} & a_{1,y} - b_{1,y} \end{pmatrix} \begin{pmatrix} b_{1,x}a_{1,y} - b_{1,y}a_{1,x} \\ b_{2,x}a_{2,y} - b_{2,y}a_{2,x} \end{pmatrix}}{\begin{vmatrix} a_{1,y} - b_{1,y} & b_{1,x} - a_{1,x} \\ a_{2,y} - b_{2,y} & b_{2,x} - a_{2,x} \end{vmatrix}}$$

Алгоритм Бентли-Оттмана

Таким образом:

- можно тем же способом эффективно сравнивать все возможные события (начала и концы отрезков и точки пересечений отрезков);

Алгоритм Бентли-Оттмана

Таким образом:

- можно тем же способом эффективно сравнивать все возможные события (начала и концы отрезков и точки пересечений отрезков);
- с помощью *Turn* можно эффективно локализовать точку при добавлении начала отрезка в статус.

Ссылки

D. Goldberg.
What every computer scientist should know about floating-point arithmetic.
ACM Comput. Surv., March, 1991.

J. Shewchuk.
Adaptive precision floating-point arithmetic and fast robust geometric predicates.
Discrete & Computational Geometry, 1996

Chee K. Yap.
Robust Geometric Computation.
Handbook of Discrete and Computational Geometry, 2004

Спасибо за внимание!