Компьютерные инструменты в образовании, 2026
№ 1: 1–??
http://ipo.spb.ru/journal

informatics.png

Научная статья

ГРАФ ВИДИМОСТИ

Баринов М. А., Павлов Д. А.

Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В. И. Ульянова (Ленина), Санкт-Петербург, Россия

1. Введение

Планирование оптимальной траектории для точечного агента на плоскости с полигональными препятствиями является фундаментальной задачей вычислительной геометрии и робототехники. Её практическая значимость охватывает такие области, как навигация мобильных роботов (в том числе беспилотных автомобилей), анализ зон видимости в геоинформационных системах, компьютерное зрение и разработка компьютерных игр. Суть задачи состоит в нахождении траектории минимальной евклидовой длины, соединяющей заданные начальную и конечную точки и не пересекающей внутренности непересекающихся многоугольных препятствий.

Классическим подходом к решению этой задачи является построение графа видимости вершин, в котором вершины препятствий соединяются ребрами, если они видят друг друга. Поиск кратчайшего пути в таком графе гарантирует оптимальность решения. Существует ряд эффективных алгоритмов построения графа видимости. В частности, алгоритм Гоша-Маунта [1] (Ghosh-Mount), предложенный в 1987 году, решает эту задачу за время $O(E + n \log n)$, где E — число ребер в итоговом графе, а n — общее число вершин препятствий. Ключевая идея этого метода заключается в использовании триангуляции пространства и особой структуры данных — «воронок» (funnels), что позволяет добавлять новые вершины и ребра в граф по одной, тратя время пропорционально только реально появляющимся связям.

Альтернативный класс методов составляют алгоритмы с произвольными углами (any-angle path planning) [5–7], которые работают непосредственно на сеточных кар-

тах, позволяя повороты под любым углом. Вот некоторые из них:

- **Theta* и Lazy Theta*** — модификации A*, использующие проверку прямой видимости между вершинами для спрямления пути;
- **ANYA** — находит оптимальные пути под любым углом, ограничивая пространство поиска узкими путями (путь, где каждое изменение направления пути «обвивается» вокруг какого-либо препятствия); рассматривая интервал точек как узел, а не как отдельную точку. Этот алгоритм ограничен двумерными сетками.
- **Field D*** — алгоритм динамического поиска пути на основе D*, использующий интерполяцию при каждом расширении вершины для нахождения почти оптимальных путей через регулярные, неравномерные сетки стоимости.

Стоит отметить, что помимо прикладных алгоритмов поиска пути, в литературе активно исследуются различные теоретические модификации графов видимости, расширяющие представление о структуре пространства. Так, в работе О'Рурка и Стрейну [8] был предложен и проанализирован **вершинно-рёберный граф видимости**, в котором учитываются не только связи между вершинами, но и видимость вершин до ребер препятствий. Авторы показали, что такая структура содержит больше информации о геометрии сцены, позволяя, например, однозначно определять тип вершин и восстанавливать деревья кратчайших путей. Хотя данная работа носит преимущественно теоретический характер, подобные результаты могут служить основой для разработки более информированных методов планирования маршрутов.

Историческая справка

Задача поиска кратчайшего пути в среде с препятствиями имеет богатую историю. Впервые понятие графа видимости было введено Нильсоном в 1968 году в контексте решения задачи планирования пути для мобильного робота Shakey [2]. Им же было доказано, что кратчайший путь между двумя точками среди многоугольных препятствий обязан проходить через их вершины.

В 1979 году Ли предложил первый эффективный алгоритм построения графа видимости со сложностью $O(n^2 \log n)$ [3, 4]. Этот алгоритм использовал метод заметающего луча и стал основополагающим для дальнейших исследований.

С развитием многоядерных процессоров в 2000-х годах акцент сместился в сторону параллельных реализаций классических алгоритмов.

2. Формальная постановка задачи

Пусть заданы:

1. **Пространство:** Евклидова плоскость $\mathbb{R}^2$.
2. **Начальная и конечная точки:** $S, T \in \mathbb{R}^2$, $S \neq T$.
3. **Множество препятствий:** $\mathcal{P} = \{P_1, P_2, \ldots, P_k\}$ — конечный набор простых многоугольников, таких что:
   - $\forall i \neq j : P_i \cap P_j = \varnothing$ (многоугольники не пересекаются);

- $S, T \notin \bigcup_{i=1}^{k} \operatorname{int}(P_i)$, где $\operatorname{int}(P_i)$ — внутренность многоугольника P_i.

Определение 1 (Допустимая траектория)**. Допустимой траекторией** называется непрерывное отображение $\phi : [0,1] \to \mathbb{R}^2$, удовлетворяющее условиям:

- $\phi(0) = S$, $\phi(1) = T$;
- $\forall t \in (0,1) : \phi(t) \notin \bigcup_{i=1}^{k} \operatorname{int}(P_i)$.

Траектория может касаться границ препятствий ∂P_i.

Определение 2 (Длина траектории)**.** Для допустимой траектории ϕ её **евклидова длина** определяется как:

$$L(\phi) = \sup \left\{ \sum_{i=1}^{n} \|\phi(t_i) - \phi(t_{i-1})\| \;\middle|\; 0 = t_0 < t_1 < \cdots < t_n = 1 \right\},$$

где $\|\cdot\|$ — евклидова норма в $\mathbb{R}^2$.

Определение 3 (Оптимальная траектория)**.** Допустимая траектория ϕ^* называется **оптимальной** (кратчайшей), если

$$L(\phi^*) = \min\{L(\phi) \mid \phi \text{ — допустимая траектория}\}.$$

3. Граф видимости

Ключевым инструментом решения задачи является дискретизация пространства с помощью графа видимости.

3.1. Построение графа

Определение 4 (Множество вершин)**.** Множество вершин графа видимости есть объединение всех вершин препятствий и начальной с конечной точками:

$$V = \{S, T\} \cup \bigcup_{i=1}^{k} V(P_i),$$

где $V(P_i)$ — множество вершин многоугольника P_i.

Определение 5 (Ребро видимости)**.** Для двух различных вершин $u, v \in V$ отрезок $[u, v]$ называется **видимым**, если

$$[u, v] \cap \left(\bigcup_{i=1}^{k} \operatorname{int}(P_i) \right) = \varnothing,$$

где $[u, v] = \{\lambda u + (1-\lambda)v \mid \lambda \in [0,1]\}$ — замкнутый отрезок.

Определение 6 (Граф видимости)**. Графом видимости** называется взвешенный неориентированный граф $G = (V, E, w)$, где:

- V — множество вершин, определённое выше;
- $E \subseteq V \times V$ — множество рёбер, такое что $(u, v) \in E$ тогда и только тогда, когда $u \neq v$ и отрезок $[u, v]$ видим;
- $w : E \to \mathbb{R}^+$ — функция весов, заданная евклидовым расстоянием: $w(u, v) = \|u - v\|_2$.

3.2. Основная теорема

Следующая теорема устанавливает фундаментальную связь между исходной непрерывной задачей и задачей поиска пути в графе. Она была доказана Нильсоном [2] и позднее уточнена Ли [3].

Теорема 1 (Основное свойство графа видимости)**.** Пусть $\phi^* : [0,1] \to \mathbb{R}^2$ — оптимальная траектория в исходной задаче, а $G = (V, E, w)$ — соответствующий граф видимости. Тогда:

(1) Существование дискретного пути: Существует путь $p = (v_0, v_1, \ldots, v_m)$ в графе G, где $v_0 = S$, $v_m = T$, такой что

$$L(\phi^*) = \sum_{j=0}^{m-1} \|v_{j+1} - v_j\|_2 = \sum_{(u,v)\in p} w(u,v).$$

(2) Структура оптимальной траектории: Оптимальная траектория ϕ^* является ломаной линией, вершины которой принадлежат множеству V: $\{v_0, v_1, \ldots, v_m\} \subseteq V$.

(3) Эквивалентность задач: Кратчайший путь в исходной задаче соответствует кратчайшему пути в графе G:

$$\min_{\phi \text{ допустимая}} L(\phi) = \min_{p \text{ путь в } G \text{ из } S \text{ в } T} \sum_{(u,v)\in p} w(u,v).$$

3.3. Алгоритмическое следствие

На основании теоремы поиск кратчайшей траектории в непрерывной области сводится к следующей двухэтапной процедуре:

1. **Построение графа видимости** $G = (V, E, w)$ для заданных препятствий $\mathcal{P}$ и точек S, T.
2. **Поиск кратчайшего пути** в графе G из вершины S в вершину T с использованием стандартных алгоритмов теории графов, таких как алгоритм Дейкстры или A*.

Сложность решения в первую очередь определяется построением графа, которое в наивной реализации требует $O(|V|^2)$ проверок видимости для каждой пары вершин, где $|V| = 2 + \sum_{i=1}^{k} |V(P_i)|$.

4. Построение графа видимости

Для начала примем, что все препятствия обходятся против часовой стрелки, так чтобы внутренность каждого препятствия находилась слева от направления обхода.

4.1. Наивный подход

Наивная реализация процедуры построения графа видимости $G = (V, E, w)$ основана на полном переборе всех возможных пар вершин и проверке видимости соответствующего отрезка.

Шаг 1. Формируется множество вершин $V = \{S, T\} \cup \bigcup_{i=1}^{k} V(P_i)$.

Шаг 2. Для каждой упорядоченной пары вершин $(u, v) \in V \times V$, где $u \neq v$, проверяется условие видимости отрезка $[u, v]$.

Шаг 3. Если отрезок $[u, v]$ видим, то в множество рёбер E добавляется неориентированное ребро (u, v) с весом $w(u, v) = \|u - v\|_2$.

Сложность данного подхода определяется количеством пар вершин, равным $O(|V|^2)$, и стоимостью проверки видимости одного отрезка, которая в наивной реализации составляет $O(n)$, где n — общее количество рёбер во всех препятствиях. Таким образом, итоговая сложность составляет $O(|V|^2 \cdot n)$, что делает метод неэффективным для сцен со сложной геометрией.

4.2. Оптимальный алгоритм на основе заметающего луча (Алгоритм Ли)

Для эффективного построения графа видимости применяется алгоритм, предложенный Ли, сложность которого составляет $O(|V|^2 \log |V|)$ в худшем случае. Алгоритм последовательно для каждой вершины $s \in V$ (называемой *источником*) определяет все другие видимые из неё вершины.

Идея метода: будем вращать луч вокруг источника s и отслеживать, какие рёбра препятствий пересекает этот луч в данный момент. Ближайшее к s пересечение определяет, закрыта ли очередная вершина каким-либо препятствием

Определение 7 (Заметающий луч)**.** Для источника s и текущей вершины v **заметающим лучом** $L(s, v)$ называется луч с началом в s, проходящий через v.

Определение 8 (Статус луча)**. Статусом** на текущем угле обхода называется упорядоченное множество (например, сбалансированное дерево) рёбер препятствий, которые пересекает заметающий луч $L(s, v)$. Рёбра упорядочены по возрастанию расстояния от точки s до точки пересечения с лучом.

4.2.1. Основные этапы для одного источника

Алгоритм для фиксированного источника s состоит из следующих этапов.

Этап 1. Подготовка и сортировка.

- Все вершины $v \in V \setminus \{s\}$ проецируются на полярную систему координат с полюсом в s.
- Вершины сортируются в порядке возрастания полярного угла $\theta(s, v)$. При равенстве углов приоритет отдаётся вершине, ближайшей к s.

- Для каждого ребра e препятствия, *видимого* из s (т.е. хотя бы один конец e видим из s), определяются события: “начало отрезка” и “конец отрезка”, соответствующие моментам, когда вращающийся луч L совпадает с направлением на концы e.

Этап 2. Инициализация.

- Луч L устанавливается в направлении $\theta = 0$ (например, горизонтально вправо).
- В статус заносятся все рёбра препятствий, которые пересекает луч в этом начальном положении, упорядоченные по расстоянию от s.

Этап 3. Последовательная обработка событий.

- События обрабатываются в порядке возрастания полярного угла.
- При наступлении события “вершина v”:
  - Если статус пуст, то вершина v *видима* из s, и ребро (s, v) добавляется в граф.
  - Если статус не пуст, то v видима только если отрезок $[s, v]$ не пересекается с ближайшим к s ребром из статуса (т.е. v находится ближе к s, чем точка пересечения луча $L(s, v)$ с этим ребром).
- При наступлении события “начало отрезка e”: ребро e добавляется в статус, если при его рассмотрении выполняется условие обхода *против часовой стрелки* для точек (s, начало e, конец e).
- При наступлении события “конец отрезка e”: ребро e удаляется из статуса, если при его рассмотрении выполняется условие обхода *по часовой стрелке* для точек (s, конец e, начало e).

После завершения обработки всех событий для источника s мы получаем все видимые из s вершины. Повторяя процедуру для каждого $s \in V$, строим полный граф видимости. Использование сбалансированных деревьев для хранения статуса обеспечивает сложность $O(\log |V|)$ на операцию вставки или удаления ребра, что приводит к общей сложности $O(|V|^2 \log |V|)$ для построения всего графа.

5. Оптимизация

Несмотря на эффективность алгоритма Ли, построенный граф видимости содержит значительное число ребер, которые заведомо не могут принадлежать ни одному кратчайшему пути. Выявление и исключение таких ребер на этапе построения позволяет сократить как время построения графа, так и время последующего поиска пути.

Определение 9 (Локальная конфигурация вершины)**.** Для вершины cur, принадлежащей некоторому препятствию P_i, определим ее **соседей** вдоль границы этого препятствия. Пусть $\mathrm{prev}(cur)$ и $\mathrm{next}(cur)$ — вершины, непосредственно предшествующая и следующая за cur при обходе контура многоугольника P_i. Для начальной и конечной точек S, T, не лежащих на границах препятствий, множество соседей считается пустым.

Теорема 2 (Критерий незначащего ребра)**.** Пусть $u, v \in V$ — две различные вершины графа видимости, и пусть отрезок $[u, v]$ является видимым. Обозначим через l_{uv} прямую, проходящую через точки u и v. Ребро (u, v) называется **незначащим** (*minor edge*), если для каждой из этих вершин выполняются следующие условия:

- вершина принадлежит некоторому препятствию;
- ее соседи вдоль границы этого препятствия существуют;
- эти соседи лежат по разные стороны от прямой l_{uv}.

Такое ребро не может принадлежать ни одному кратчайшему пути в графе видимости и может быть исключено из рассмотрения (рис. 1, красным цветом выделено такое ребро, а зеленым допустимые).

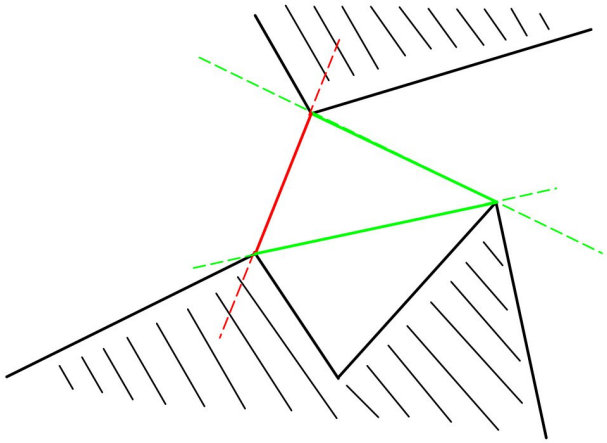

Рис. 1. Незначащее ребро

А также:

1. Можно заметить следующее: если мы имеем две вершины u и v такие, что первая из них расположена ниже второй (то есть ее y-координата меньше), тогда при обработке вершины v ребро (u, v) либо будет проведено, если оно допустимо, либо не будет. Следовательно, можно не рассматривать вершины, у которых y-координата меньше текущей.
2. Рассмотрим три последовательные вершины $\mathrm{prev}(cur)$, cur и $\mathrm{next}(cur)$. Если эти вершины образуют поворот по часовой стрелке, то вершина cur является вогнутой, и все инцидентные ей ребра в графе видимости являются незначащими согласно теореме 2 (рис. 2). Следовательно, такая вершина cur может быть полностью исключена из рассмотрения в качестве кандидата на проведение ребер.

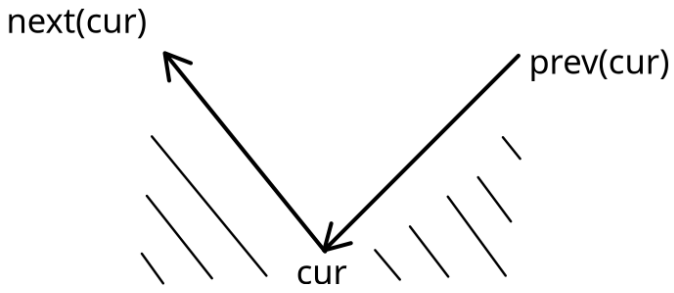

Рис. 2. Вогнутая часть препятствия

3. Если же указанные вершины образуют поворот против часовой стрелки, то вершина *cur* является выпуклой. В этом случае при обработке вершины *cur* все ребра, построенные в вершины, лежащие внутри сектора, образованного лучом из prev(*cur*) через *cur* и лучом из next(*cur*) через *cur*, будут незначащими (рис. 3). Данное множество вершин может быть исключено из проверки видимости.

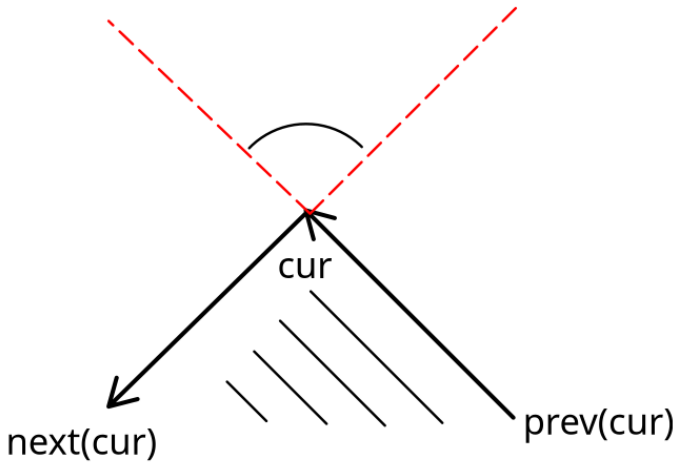

Рис. 3. Выпуклая часть препятствия

В реализации последняя оптимизация может оказаться сложнее остальных. Например, можно действовать следующим образом:

1. Запускаем стандартный алгоритм Ли для источника, но заметающий луч совершает не полный оборот, а доходит до начала сектора, образованного лучами, описанными в пункте 2.

2. Далее немного изменим алгоритм Ли:

   - Луч L устанавливается в направлении горизонтально влево (при условии, что применен п. 1).

- В статус аналогично вносятся все ребра препятствий, которые пересекает луч в этом начальном положении, упорядоченные по расстоянию от источника.
- События обрабатываются в порядке убывания полярного угла.

3. Обработка продолжается до достижения другой границы сектора.

В результате вершины, находящиеся внутри этого сектора, не обрабатываются, что позволяет существенно сократить время работы алгоритма.

Можно заметить, что обработка вершин происходит независимо друг от друга, так как для каждой вершины рассматриваются вершины только выше ее самой (те, у которых y-координата больше). То есть подход, описанный выше, можно распараллелить:

Список отсортированных вершин можно обрабатывать параллельно. Для этого запустим несколько потоков, каждый из которых получит свою вершину. Чтобы потоки не мешали друг другу, дадим им задания не все сразу, а порциями: сначала раздадим вершины всем потокам, ждем, пока они закончат, потом берем следующую группу вершин и повторяем процедуру. Такой подход позволяет эффективно использовать все ядра процессора и значительно ускоряет работу алгоритма.

6. Хранение графа

В задачах, связанных с графами, важную роль играет выбор способа их хранения. Наиболее простым, но не самым оптимальным с точки зрения производительности подходом, является хранение для каждой вершины контейнера с инцидентными ей рёбрами или соседними вершинами. Такой подход часто используется в различных проектах и может выглядеть следующим образом:

```
struct Vertex
{
    std::set<Edge*> edges;
};

struct Edge
{
    Vertex* a;
    Vertex* b;
};
```

Однако данный способ уступает по эффективности альтернативным решениям, предложенным далее (см. таблицу 1).

7. Результаты тестирования производительности

В качестве результатов сравним время работы наивного алгоритма и оптимизированного алгоритма Ли.

7.1. Тестовые данные

Тестирование проводилось на планарном графе, изображенном на рисунке 4. Характеристики графа:

- количество вершин: $n = 6578$;
- количество ребер: $m = 6559$.

Рис. 4. Исходный планарный граф

После применения алгоритма построения графа видимости количество ребер увеличилось до 12668. Результирующий граф видимости представлен на рисунке 5.

Рис. 5. Построенный граф видимости

7.2. Характеристики тестового стенда

Тестирование проводилось на машине со следующими характеристиками:

- процессор: AMD Ryzen 5 5600X 6-Core Processor;
- тактовая частота: базовая 2.2 ГГц, максимальная 4.65 ГГц;
- оперативная память: 32 ГБ;
- операционная система: Ubuntu 24.04.4 LTS.

7.3. Результаты замеров времени

В таблице 1 представлены результаты замеров времени работы различных реализаций алгоритмов.

Алгоритм	**Время выполнения, с**
Наивный алгоритм ($O(n^2 \cdot m)$) с плохим хранением графа	33,164
Наивный алгоритм ($O(n^2 \cdot m)$) с хорошим хранением графа	26,074
Алгоритм Ли с оптимизациями	0,963
Алгоритм Ли с оптимизациями и распараллеливанием	0,340

Таблица 1. Сравнение времени работы алгоритмов

Как видно из полученных результатов, оптимизированный алгоритм Ли показывает значительный прирост производительности по сравнению с наивной реализацией. Дополнительное распараллеливание вычислений позволяет ускорить работу еще в 2.5 раза.

Заключение

В результате проделанной работы удалось значительно ускорить алгоритм Ли для построения графа видимости. Были реализованы и протестированы наивный алгоритм, оптимизированная версия алгоритма Ли, а также его параллельная модификация. Эксперименты на сцене с 6578 вершинами показали, что оптимизированный алгоритм работает более чем в 25 раз быстрее наивного, а распараллеливание на 12 потоков даёт дополнительное ускорение в 2.5 раза.

Полученные результаты могут быть использованы в системах планирования пути, работающих в реальном времени, а также в научных исследованиях, связанных с вычислительной геометрией.

Список литературы

1. Ghosh S. K., Mount D. M. An Output-Sensitive Algorithm for Computing Visibility Graphs // Proceedings of the 28th Annual Symposium on Foundations of Computer Science (FOCS'87). – IEEE, 1987. – P. 11–19.
2. Nilsson N. J. A mobile automaton: An application of artificial intelligence techniques // Proceedings of the 1st International Joint Conference on Artificial Intelligence (IJCAI'69). – 1969. – P. 509–520.
3. Lee D. T. Proximity and reachability in the plane : Ph.D. Thesis / University of Illinois at Urbana-Champaign. – 1978.
4. D. Coleman. Lee's O(n^2 log n) visibility graph algorithm: Implementation and analysis. Technical report, University of Colorado at Boulder, 2012.
5. A. Nash, K. Daniel, S. Koenig, and A. Felner. Theta*: Any-angle path planning on grids. In *Proc. of the AAAI Conference on Artificial Intelligence*, 2007.
6. A. Nash, S. Koenig, and C. Tovey. Lazy Theta*: Any-angle path planning and path length analysis in 3D. In *Proc. of the AAAI Conference on Artificial Intelligence*, 2010.
7. D. Harabor and A. Grastien. An optimal any-angle pathfinding algorithm. In *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*, 2016.
8. J. O'Rourke and I. Streinu. The vertex-edge visibility graph of a polygon. *Computational Geometry: Theory and Applications*, 10(2):105–120, 1998.