ПОИСК ОПТИМАЛЬНОГО ПУТИ НА МНОЖЕСТВЕ ПРЕГРАД В КОМПЬТЕРНЫХ ИГРАХ

Василий Терещенко, Денис Янчик, Дмитрий Пустовойтов
Факультет кибернетики
Киевский национальный университет имени Тараса Шевченко, Киев, Украина
v_ter@ukr.net, razor.den@gmail.com, dmytro.pustovoytov@gmail.com

Аннотация

В работе представлен алгоритм поиска оптимального пути на плоскости с учетом преград в виде простых не пересекающихся многоугольников. Алгоритм состоит из множества эффективных алгоритмов, которые в совокупности образуют единый оптимальный алгоритм. Общая временная сложность алгоритма *O(nlogn)*, объем используемой памяти *O(n),* где *n* – суммарное количество всех вершин многоугольников, из которых формулируется область поиска.

Ключевые слова: *Оптимальный путь, Преграды, Монотонный многоугольник,Цепь, Компьютерные игры.*

1. ВВЕДЕНИЕ

Постановка проблемы. В современных компьютерных играх-стратегиях разработчики часто решают такую проблему: провести некоторые отряды юнитов между двумя пунктами по пересеченной местности кратчайшим путем. Построение эффективного алгоритма решения этой проблемы позволило бы оптимизировать использование вычислительных ресурсов компьютера и создавать более сложные игры с препятствиями любой формы и размеров. По сути, такая задача сводится к нахождению кратчайшего пути на графах. На сегодняшний день, одним из самых эффективных алгоритмов поиска пути по графу, является A* - алгоритм [1] и его оптимизации [1, 2]. Большинство предложенных оптимизаций этого алгоритма связаны с поиском решения проблемы быстрой и оптимальной обработки большого количества данных и преобразования их к виду, приемлемому для A* - алгоритма. В особенности, такая проблема возникает, если игровое пространство непрерывно. При этом позиции объектов и препятствий сохранены в виде непрерывных значений и должны быть настолько точно представлены, как и разрешение экрана. Примером решения этой проблемы могут быть подходы, использующиеся для мобильных роботов. Одним из таких подходов есть сведение непрерывного пространства к нескольким дискретным вариантам [2]. Способы дискретизации пространства включают:

- нанесение ячеистой сетки на поверхность пространства поиска;

- выделение критических точек, расположенных, в основном, в близи вершин препятствий обхода;

- разбиение на выпуклые полигоны пространства, не занимаемого полигональными препятствиями; промежуточными точками могут быть центры полигонов или точки на границах полигонов [3];

- разбиение на квадраты, где каждый квадрат, не являющийся однородным, разделяется на четыре меньших квадрата, центры которых используются для поиска пути;

- пространство между смежными препятствиями рассматривается как цилиндр, форма которого изменяется вдоль его оси. Вычисляется ось, проходящая через пространство между двумя смежными препятствиями (включая стены), и эти оси используются для поиска пути.

- представления препятствия в виде потенциального поля [4], сила которого обратно пропорциональна расстоянию до него. Так же существует однородная сила притяжения к цели. Через близкие постоянные интервалы времени вычисляется сумма притягивающих и отталкивающих векторов и объект передвигается в этом направлении. Проблема этого подхода состоит в попадании объекта в локальный минимум.

Цель работы - построение общего эффективного алгоритма поиска кратчайшего пути, на множестве преград в виде простых многоугольников, оптимизирующего современные компьютерные игры по количеству юнитов в играх жанра “стратегии”, а также по сложности и разнообразию препятствий в играх жанра “екшен”.

В работе предложен новый подход решения рассматриваемой задачи, который позволяет реализовывать игры на качественно новом уровне.

2. ОСНОВНАЯ ЧАСТЬ

В основе представленного алгоритма лежит идея интеграции нескольких эффективных методов решения задач вычислительной геометрии и геометрического моделирования для достижения максимально быстрого поиска кратчайшего пути на плоскости. При этом используется минимальный объем памяти для хранения структур данных.

Постановка задачи. *Пусть задано две точки – А (стартовая) и Б (конечная). Область поиска (внешний мир) представлена в виде простого многоугольника, а преграды внутри рассматриваемого мира описаны в виде самонепересекающихся и не пересекающих внешнюю границу мира простых многоугольников, рис. 1. Необходимо найти кратчайший путь между точками А и В.*

В случае самопересечения или пересечения между собой следует решить задачу о разделении таких многоугольников на множество простых самонепересекающихся и непересекающихся между собой многоугольников, что не является целью данной статьи.

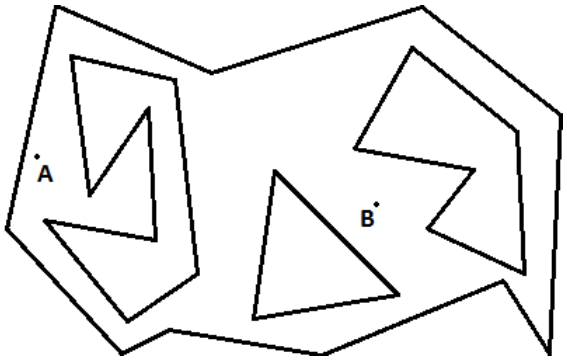

Рисунок 1: Область поиска – простой многоугольник.

Пусть многоугольники представляются и хранятся в виде циклически упорядоченной последовательности точек (вершин). Объем памяти, необходимый на хранение такой структуры *O(n)*.

Чтобы решить задачу, разобьем ее на пять подзадач: регуляризация графа, выделение монотонных многоугольников, триангуляция монотонных многоугольников, локализация точки, A* - алгоритм. Для решения этих подзадач будем использовать известные эффективные алгоритмы. Учитывая то, что эти алгоритмы детально описаны, рассмотрим лишь вкратце каждый их них, на примере, рис.1.

2.1 Регуляризация графа

Регулярность графа рассматривается относительно некоторой прямой *l*. Не ограничивая общности, пусть это будет ось *OX*. Сориентируем граф (рис.1.) по возрастанию в направлении оси *OX*. Тогда вершину *v* будем называть *регулярной*, если множества входящих и исходящих из этой вершины ребер одновременно не пустые. Вершина *v* называется *не регулярной*, если хотя бы одно из множеств, входящих или исходящих из нее ребер, пустое.

Граф называется *регулярным* относительно некой прямой (например, оси *OY* или *OX*), если все его вершины, кроме первой и последней, регулярны. Если же хотя бы одна из вершин (кроме первой и последней) не регулярна, граф называется *не регулярным*. *Регуляризация* графа – процесс преобразования не регулярного графа в регулярный.

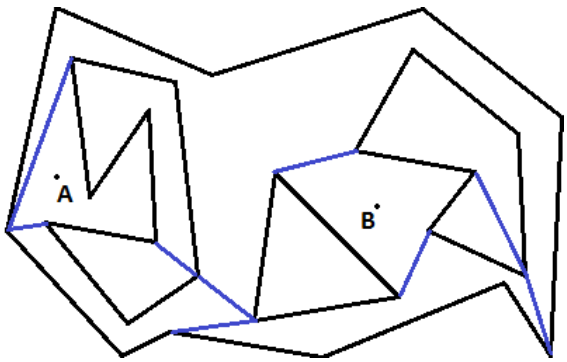

Рисунок 2: Область поиска после применения алгоритма регуляризации.

Граф на рис. 1 не регулярен относительно оси *OX* и его необходимо регуляризировать. Для этого используем алгоритм плоского заметания [5], в результате чего получим область поиска в виде регулярного графа, рис. 2.

2.2 Выделение монотонных многоугольников методом цепей

Следующий шаг алгоритма - разбиение полученной области поиска (рис. 2) на монотонные многоугольники. Для этого наиболее подходящим является эффективный алгоритм монотонных цепей (определения монотонной цепи и монотонного многоугольника можно найти в [5]), предложенный Ли и Препаратой [6] и усовершенствованный в работе [7], который применяется для решения задачи локализации точки на плоскости. Согласно этому алгоритму, за три линейных прохода по графу (рис. 2) мы получим сбалансированный взвешенный граф, который расцепляется на полное множество монотонных, относительно оси OX, цепей, рис. 3.

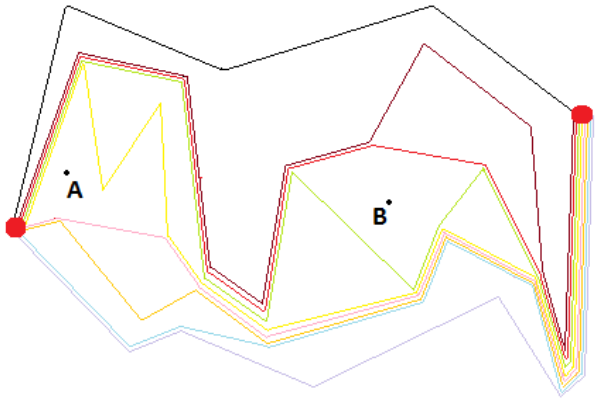

Рисунок 3: Выделенные цепи.

На основе этого множества строится структура данных – взвешенное бинарное дерево монотонных цепей. Эта структура данных позволяет нам за линейное время произвести триангуляцию графа и за *O(logn)* локализировать точку.

2.3 Триангуляция монотонных многоугольников

В этой задаче необходимо триангулировать монотонные многоугольники, образованные множеством последовательных монотонных цепей графа на рис.3. Полигон называется *монотонным*, если он состоит из двух (верхней и нижней), монотонных относительно одной и той же прямой *l,* цепей [5].

При выделении многоугольников методом цепей все полученные многоугольники являются монотонными так, как все выделенные цепи – монотонные относительно оси OX (вертикальное заметание слева направо обеспечивает монотонность), а все многоугольники образованы двумя последовательными монотонными цепями (монотонность по условию). Более детально это описано в [8], [9].

Алгоритм триангуляции монотонных многоугольников. Переименуем вершины нашего монотонного многоугольника в $v_{1,}\ v_{2,\ \ldots\ ,}\ v_n$ в порядке возрастание координаты абсцисс (именно в таком порядке наш алгоритм будет обрабатывать вершины). Алгоритм хранит стек s вершин, которые были проверены, но полностью не обработаны. Алгоритм формирует последовательность монотонных полигонов $p = p_1, p_2, \ldots, p_n = 0$. Полигон p_i, как результат обработки вершины v_i, получается путем отсечения нуля или нескольких треугольников от предыдущего полигона. Алгоритм завершает работу с пустым полигоном, а множество треугольников, полученное в процессе обработки, представляет собой триангуляцию исходного полигона p. Пусть $s_{1,}\ s_{2,\ \ldots\ ,}\ s_n$ вершины со стека в порядке возрастания снизу вверх, тогда на каждом шаге возможно три варианта их расположения:

1. $s_{1,}\ s_{2,\ \ldots\ ,}\ s_t$ упорядочены до возрастанию координаты x и содержат каждую из вершин полигона p_{i-1}, расположенную справа он s_1 и слева от s_t.
2. $s_{1,}\ s_{2,\ \ldots\ ,}\ s_t$ являются последовательными вершинами либо в верхней, либо в нижней цепочках полигона p_{i-1}.
3. $s_{1,}\ s_{2,\ \ldots\ ,}\ s_n$ являются вогнутыми вершинами полигона p_{i-1} (внутренний угол каждой из них более 180^o).

Следовательно, последующая вершина v_i в обработке может быть в трех соотношениях с вершинами $s_{1,}\ s_{2,\ \ldots\ ,}\ s_t$, рис 4:

a. v_i соседняя с s_t, но не с s_1;
b. v_i соседняя с s_1, но не с s_t;
c. v_i соседняя и с s_1, и с s_t;

Ядро полигона – подмножество его точек, из которых виден весь полигон (ядро выпуклого полигона он сам). Полигон называется *веерообразным*, если ядро содержит одну или более вершин – *корней* полигона. Соответственно действия в каждом из трех случаях разные.

В случае *a*: пока $\angle v_i\ s_t\ s_{t-1} < 180^o$, отсекаем полигон $v_i\ s_t\ s_{t-1}$ и уменьшаем t. Когда угол стал больше 180^o либо t стало равным 1, заносим v_i в стек, рис 4 *a*.

В случае *b:* отсекаем полигон $v_i\ s_t\ s_{t-1\ldots}\ s_1$ – он является веерообразным с узлом в точке v_i, триангулируем его. Заносим в стек s_t, после - v_i, рис. 4 *b*.

В случае *c*: v_i является v_n и полигон $v_i\ s_t\ s_{t-1\ldots}\ s_1$ является веерообразным с углом в точке v_n. Выполняем триангуляцию и завершаем алгоритм, рис 4 *c*.

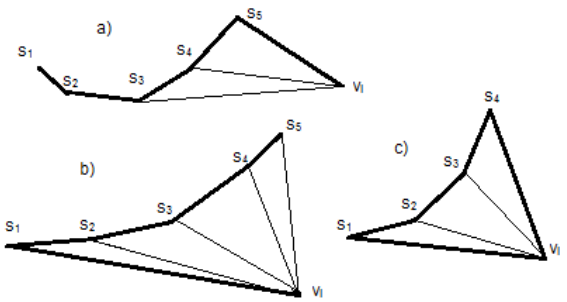

Рисунок 4: Три случая обработки следующей вершины v_i.

В результате получим триангулированный граф, рис. 5. Порядок выполнения шагов триангуляции сверху вниз, слева направо. Более детальное описание алгоритма представлено в работах [5, 9, 10].

2.4 Локализация точки методом цепей

Для локализации точек A и B будем использовать метод цепей [6] и структуру данных (бинарное дерево цепей) построенную в разделе 2.2. Сделать это достаточно легко, учитывая то, что у нас имеется планарный граф. Запускаем метод цепей для уже триангулированной области поиска. Особенности хранения данных в памяти позволяют бинарным поиском найти, между какими 2-мя цепями находиться наша точка. После этого не составит труда вычислить, в каком треугольнике находиться точка. После этого можно приступать к применению A* - алгоритма.

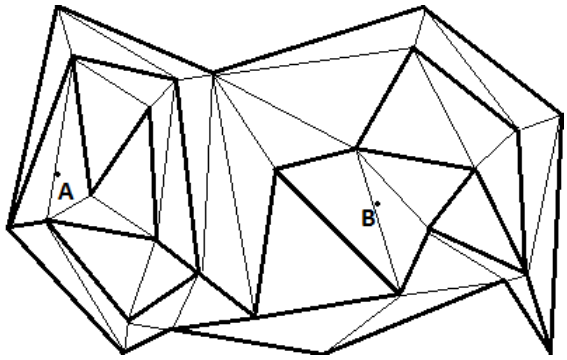

Рисунок 5: Триангуляция области поиска.

2.5 A* алгоритм

A*- алгоритм [1, 2 11] пошагово просматривает все пути, ведущие от начальной вершины в конечную, пока не найдет минимальный. Как и все алгоритмы поиска, он просматривает сначала те маршруты, которые «кажутся» ведущими к цели. От жадного алгоритма (который тоже является алгоритмом поиска по первому лучшему совпадению) его отличает то, что при выборе вершины он учитывает, помимо прочего, весь пройденный до неё путь (составляющая $g(x)$) – это стоимость пути от начальной вершины, а не от предыдущей, как в жадном алгоритме) [1].

В начале работы рассматриваться узлы, смежные с начальной точкой; выбирается тот из них, который имеет минимальное значение $f(x)$, после чего этот узел раскрывается. На каждом этапе алгоритм оперирует с множеством путей из начальной точки до всех ещё не раскрытых (листовых) вершин графа («множеством частных решений»), которые размещается в очереди с приоритетом. Приоритет пути определяется по значению $f(x) = g(x) + h(x)$. Алгоритм продолжает свою работу до тех пор, пока значение $f(x)$ целевой вершины не окажется меньшим, чем любое значение в очереди (либо пока всё дерево не будет просмотрено). Из множества решений выбирается решение с наименьшей стоимостью. Чем меньше эвристика $f(x)$, тем больше приоритет (поэтому для реализации очереди можно использовать сортирующие деревья).

Алгоритм A* обходит при этом минимальное количество вершин, благодаря тому, что он работает с «оптимистичной» оценкой через вершину. Оптимистичной в том смысле, что, если он пойдёт через эту вершину, то у алгоритма «есть шанс», что реальная стоимость результата будет равна этой оценке, но никак не меньше.

Когда A* завершает поиск, то он, согласно определению, нашёл путь, истинная стоимость которого меньше, чем оценка стоимости любого пути через любой открытый узел. Но поскольку эти оценки являются оптимистичными, соответствующие узлы можно без сомнений отбросить. Иначе говоря, A* никогда не упустит возможности минимизировать длину пути, и потому является допустимым.

Предположим теперь, что некий алгоритм *B* выдал в качестве результата путь, длина которого больше оценки стоимости пути через некоторую вершину. На основании эвристической информации, для алгоритма *B* нельзя исключить возможность, что этот путь имел и меньшую реальную длину, чем полученный результат. Соответственно, пока алгоритм *B* просмотрел меньше вершин, чем A*, он не будет допустимым. Итак, A* проходит наименьшее количество вершин графа среди допустимых алгоритмов, использующих такую же точную (или менее точную) эвристику. Более подробную информацию, с различными модификациями и оптимизациями A*- алгоритма можно найти в [11].

3. АНАЛИЗ СЛОЖНОСТИ АЛГОРИТМОВ

Общая сложность алгоритма решения задачи поиска оптимального пути на плоскости с учетом преград в виде простых не пересекающихся многоугольников определяется суммой сложностей алгоритмов пяти подзадач.

1. Время, потраченное на регуляризацию n – вершинного планарного графа составляет $O(n\log n)$ с использованием $O(n)$ памяти [5].

2. Общее время алгоритма выделения монотонных многоугольников в заданном графе равно $O(n\log n)$. При этом, $O(n\log n)$ времени уходит на предобработку (сортировка ребер графа за и против часовой стрелки), после инициализация, 1-ый и 2-ой проходы по $O(n)$ каждый – мы бываем в каждой вершине всего 1 раз. $O(n)$ - сложность выделения многоугольников из последовательности цепей.

3. На триангуляцию монотонных многоугольников уходит $O(n)$ времени. Задаваемый на входе полигон содержит n вершин. Заметим, что при каждой итерации двух внутренних циклов while из стека исключается одна вершина. Однако каждая вершина записывается в стек только однажды (при первой ее обработке) и, следовательно, может быть выбрана из стека тоже только однажды. Поскольку алгоритм выполняет $O(n)$ операций со стеком одинаковой длительности и затрачивает одинаковое время между двумя такими последовательными операциями, то время работы программы пропорционально $O(n)$. Нижняя граница определяется тем, что должна быть обработана каждая из n вершин [12]. Кроме того, в работе [10] показано, что триангуляцию можно оптимизировать до сложности $O(n\log(\log n))$. А Чазелле показал [4], что простой многоугольник может быть триангулирован за линейное время.

4. Локализацию точки в n-вершинном планарном подразбиении можно реализовать методом цепей [6] за время $O(\log^2 n)$ с использованием $O(n)$ памяти при затратах $O(n\log n)$ времени на предобработку. Учитывая оптимизацию нашего алгоритма под метод цепей, сложность поиска составляет порядка $O(\log(2\log n))$.

5. Временная сложность алгоритма A* зависит от эвристики. В худшем случае, число вершин, исследуемых алгоритмом, растёт экспоненциально по сравнению с длиной оптимального пути, но сложность становиться полиномиальной, когда эвристика удовлетворяет следующему условию:

$$|h(x) - h^*(x)| \le O(\log h^*(x))$$

Где h^* - оптимальная эвристика, то есть точная оценка расстояния из вершин x к цели. Другими словами, ошибка $h(x)$ не должна расти быстрее, чем логарифм от оптимальной эвристики, [1].

4. ПРАКТИЧЕСКАЯ ЧАСТЬ

На основе выше изложенной теории нами была разработана практическая реализация данного алгоритма. На Рис. 6 приведен график зависимости скорости работы программы от суммарного количества вершин всех преград.

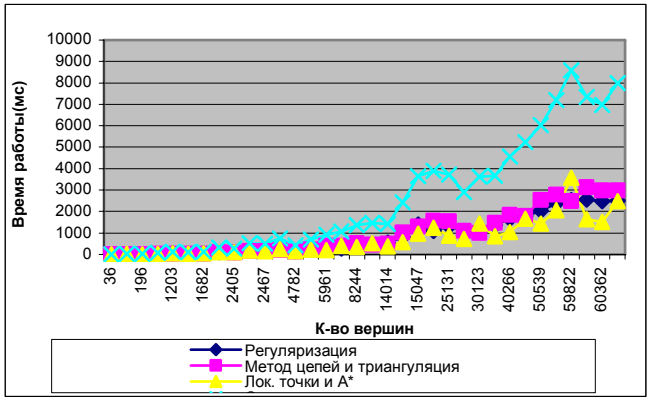

Рисунок 6: Результаты работы алгоритма.

С графика видно, что время работы всех алгоритмов в отдельности имеют один порядок сложности, причем он не является квадратическим. Таким образом, можно утверждать, что приведенные теоретические оценки производительности имеют место и на практике.

5. ЗАКЛЮЧЕНИЕ

В статье рассматривается новый алгоритм поиска оптимального пути на множестве преград. Представленный алгоритм реализуется с помощью решения пяти подзадач за оптимальное время $O(n\log n)$ (сложность всех 5-ых подзадач составляет не более $O(n\log n)$), используя минимальный объем памяти ($O(n)$), для хранения структур данных. Предложенный обобщенный алгоритм можно применять при создании компьютерных игр жанра “стратегия”, позволяя моделировать динамическое изменение области поиска.

6. СПИСОК ЛИТЕРАТУРЫ

[1] Lester, Patrick. "A* Pathfinding for Beginners". 21 Oct. 2006 http://www.policyalmanac.org/games/aStarTutorial.htm.

[2] Bryan Stout. Smart move: Intelligent path-finding. Online at gamasutra.com/view/feature/3317/smart move intelligent.php, 1999. 8.

[3] Y.-J. Chiang and R. Tamassia. Optimal shortest path and minimum-link path queries between two convex polygons inside a simple polygonal obstacle. Internat. J. Comput. Geom. Appl., 7, 1997, pp. 85-121.

[4] Chazelle B. Triangulating a Simple Polygon in Linear Time. Discrete Comput. Geom. 6 (1991), 485-524.

[5] Препарата Ф., Шеймос М. Вычислительная геометрия: Введение / Пер. с англ. М.: Мир, 1989.

[6] D. T. Lee and F.P. Preparata. Location of a point in a planar subdivision and its applications. SIAM Journal on computing 6(3), 1977, pp. 594-606.

[7] E. Edelsbrunner, L. J. Guibas and J. Stolfi. Optimal point location in a monotone subdivision. SIAM J. Computing. 15(2) , 1986, pp. 317 – 340.

[8] Скворцов А.В., Костюк Ю.Л. Применение триангуляции для решения задач вычислительной геометрии // Геоинформатика: Теория и практика. Вып. 1. Томск: Изд-во Томск. ун-та, 1998. С. 127–138.

[9] Gilbert P.N. New results on planar triangulations. Tech. Rep. ACT-15, Coord. Sci. Lab., University of Illinois at Urbana, July 1979.

[10] Tarjan R.E., Van Wyk C. J. An D(ηloglogη)-time algorithm for triangulating a simple polygon, SIAM J. Comput. 17 (1988), pp. 143-178.

[11] Björnsson Y., Enzenberger M., Holte R. C., Schaeffer J.. Fringe Search: Beating A* at Pathfinding on Game Maps. In Proc. of the 2005 IEEE Symposium on Computational Intelligence and Games. Colchester, Essex, UK, 2005, pp. 125-132.

[12] М.Ласло. Вычислительная геометрия и компьютерная графика на С++ : Пер. с англ. В. Львова. - М.: Бином, 1977.

About the author

Vasyl Tereshchenko is an associate professor at National Taras Shevchenko University of Kyiv, Faculty of Cybernetics. His contact email is v_ter@ukr.net.

Denis Yanchik is a student at National Taras Shevchenko University of Kyiv, Faculty of Cybernetics. His contact email is razor.den@gmail.com

Dmitro Pustovoytov is a student at National Taras Shevchenko University of Kyiv, Faculty of Cybernetics. His contact email is dmytro.pustovoytov@gmail.com