Path Tracing

© Селифонов Евгений, Тихомиров Андрей
(СПбГУ ИТМО, 2007)
© http://rain.ifmo.ru/cat

Оглавление

I. Введение

I.1. Определение трассировки

- ***Трассировка*** – задача о нахождении пути из одной точки в другую по местности, содержащей непроходимые или труднопроходимые препятствия
- **Не всегда** трассировка ставит перед собой цель найти ***кратчайший*** путь; зачастую это просто невозможно
- Иногда рассматриваются задачи с различной трудностью прохождения препятствий

I.2. Основные классы задач

- Все задачи трассировки можно разделить на 3 категории:
  - навигация в **известных** *дискретных* средах;
  - навигация в **известных** *непрерывных* средах;
  - навигация в **неизвестных** средах.

I.2. Основные классы задач

- Задачи трассировки можно разделить на 2 большие категории:
  - Задачи **строительной** трассировки:
    - Препятствия – строения, естественные (горы, озера) и искусственные препятствия (запретная зона)
  - Задачи **электронной** трассировки:
    - Препятствия – те же, что и у строительной + ранее проведенные трассы
- Мы будем рассматривать задачи строительной трассировки

I.3. Некоторые понятия

- **Точкой старта** назовем ту точку, из которой мы должны провести маршрут
- **Точкой цели** назовем ту точку, в которую мы должны провести маршрут
- **Роботом** назовем движущийся по маршруту объект

II. Трассировка в известных дискретных средах

II.1. Введение

- В данном разделе рассматривается простейшая задача трассировки: обход двумерных лабиринтов
- Должна быть задана растровая карта в виде двумерной плоскости, разделенной на клетки (квадратные, треугольные или гексагональные), с отмеченными стартом, целью и препятствиями

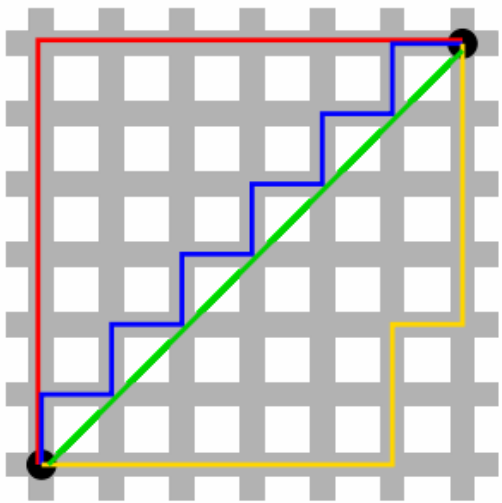
II.2. Taxicab Geometry

- Часто используется отличная от евклидовой геометрия, называемая Taxicab Geometry
- Расстояние, используемое в данной геометрии и называемое расстоянием Манхеттена (Manhattan Distance), между точками с координатами (x_1,y_1) и (x_2,y_2) вычисляется по формуле:

$$|x_1 - x_2| + |y_1 - y_2|$$

II.3. Manhattan Distance

- Легко увидеть, что расстояние Манхеттена (красная, синяя и желтая линии) отличается от евклидового (зеленая линия) (в данном примере - в $\sqrt{2}$ раз)



II.4. Выбор стратегии

- Мы можем пойти двумя путями:
  - Прокладывать путь «на ходу», игнорируя препятствия до столкновения с ними
  - Заранее спланировать путь до начала перемещения

II.5.1. Выбор стратегии: первый путь

- Общий алгоритм будет следующим:
  - Пока цель не достигнута:
    - Выбрать направление для движения к цели
    - Если это направление свободно для движения:
      - Двигаться туда
    - Иначе:
      - Выбрать другое направление в соответствии со стратегией обхода

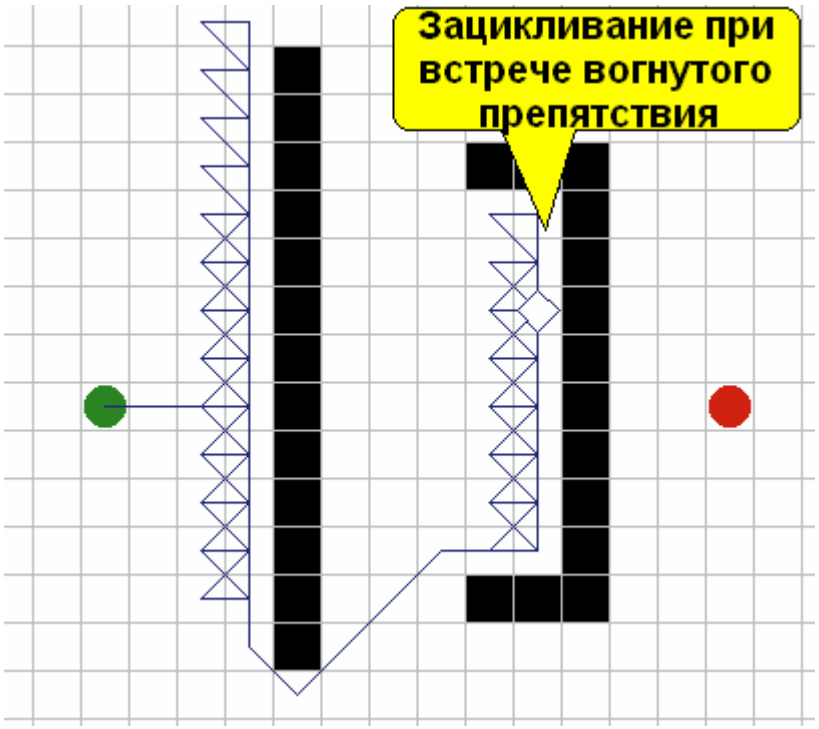
II.5.2. Первый путь

- Некоторые примеры алгоритмов:
  - Перемещение в случайном направлении
    - Делаем небольшое смещение в случайном направлении при встрече препятствия
  - Трассировка вокруг препятствия
    - При встрече препятствия идем по его контуру до некоторого момента, определяемого эвристикой
  - Надежная трассировка
    - Идентично алгоритмам семейства Bug, которые мы рассмотрим позже для более общего случая

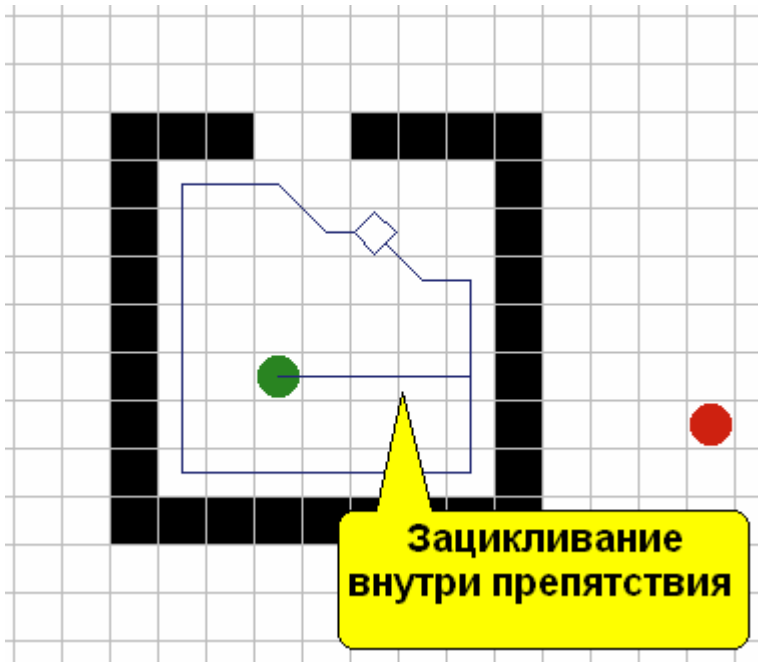
II.5.3. Первый путь: преимущества и недостатки

- Преимущества:
  - Простота: все, что необходимо знать, - это относительное положение робота к цели и признак блокирования препятствием
  - Малое потребление памяти
- Недостатки:
  - Ненадежность (кроме последнего алгоритма)
  - Неработоспособность во взвешенных средах

II.5.4. Первый путь: недостатки



Недостатки алгоритма перемещения в случайном направлении



Недостатки алгоритма перемещения вокруг препятствия

II.6. Выбор стратегии: второй путь

- Заметим, что наш лабиринт можно представить в виде графа, соответственно, можно воспользоваться некоторыми известными алгоритмами на графах для предварительного планирования пути
- Под словами «конструируем путь» будем понимать создание списка вершин, которые необходимо пройти от старта к цели
- Для конструирования пути введем указатель **Parent[n]** на ту вершину, из которой мы приходим в n; т.о. добавим родителя целевой вершины в наш список, затем добавим родителя той вершины и т.д. до стартовой точки

II.7.1. Поиск в ширину

- Простейшим алгоритмом нахождения пути является поиск в ширину (другое название - волновой алгоритм)
- Мы запускаем волну из стартовой точки, которая постепенно заполняет пространство, в итоге доходя до целевой точки

II.7.2. Поиск в ширину: алгоритм

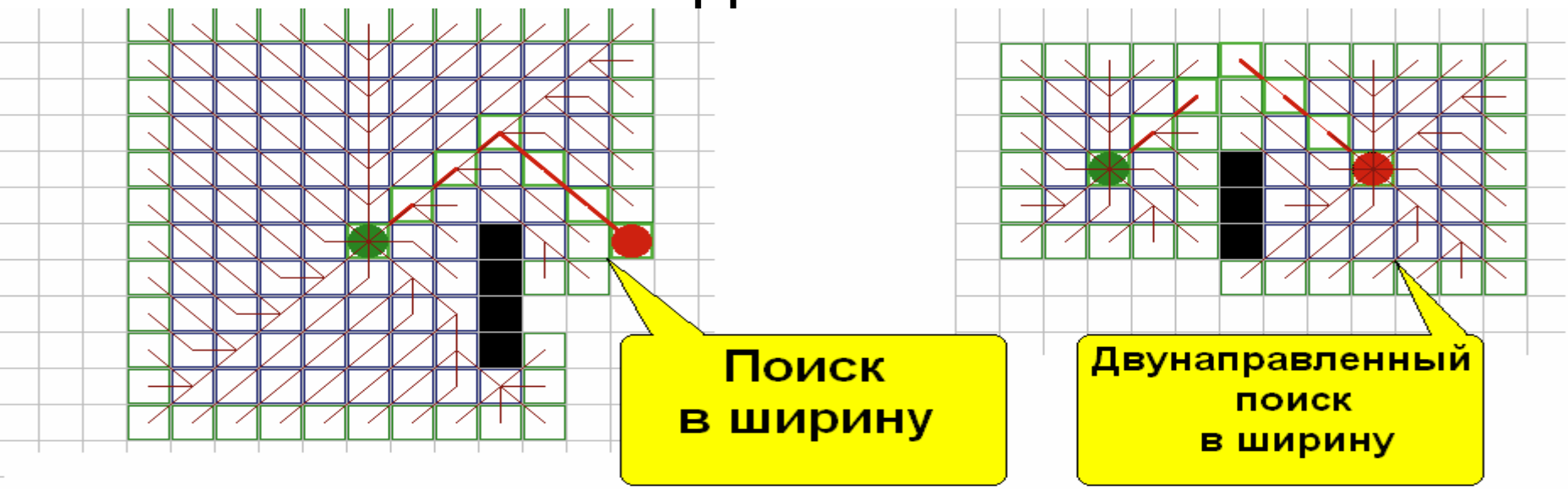
- Создаем **очередь** Open
- Open ← **Start** (кладем в очередь)
- **Пока** [Open не пуста]
  - Open → **N** (извлекаем из очереди)
  - **Если** [**N** = **Goal**]
    - Конструируем путь и выходим
  - **Для каждого** [**M** (непосещенного соседа узла **N**)]
    - **Parent[M]** ← **N**
    - Open ← **M**
- Путь не найден, выходим

II.7.3. Поиск в ширину: преимущества и недостатки

- Преимущества:
  - Простота реализации
  - Всегда находит кратчайший путь при условии равенства весов
- Недостатки:
  - Поиск идет равномерно во всех направлениях, вместо того, чтобы быть направленным к цели
  - Не всегда все шаги равны (например, диагональные шаги должны быть длиннее ортогональных)

II.7.4. Двунаправленный поиск в ширину

- Запускается две волны: из стартовой и целевой точек
- Алгоритм работает до встречи двух волновых фронтов

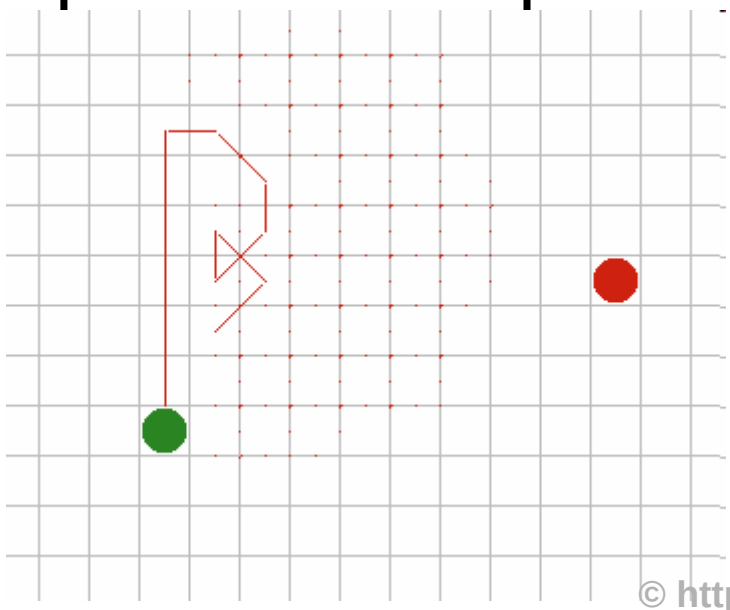


II.8.1. Поиск в глубину

- Алгоритм противоположен поиску в ширину
- Вместо очереди используется стек
- Можно обойтись без списка Open за счет рекурсии

II.8.2. Поиск в глубину: недостатки

- Часто получаются бессмысленные пути (см. рисунок)
- Длительное время работы
- Игнорирование направления к цели



II.8.3. Поиск в глубину: оптимизации

- На каждую ячейку будем добавлять метку с длиной найденного к ней кратчайшего пути; больше не будем посещать эту ячейку, пока к ней не будет найден путь с меньшей стоимостью
- Будем выбирать сначала соседей, которые находятся ближе к цели
- **Алгоритм последовательных приближений**: будем делать остановку на определенной глубине, сначала равной расстоянию от старта до цели и постепенно увеличивающейся

II.9.1. Алгоритм Дейкстры

- Один из наиболее мощных алгоритмов для поиска кратчайшего пути в графах, ребра которых имеют различный вес
- За **g(n)** обозначим стоимость пути от старта до узла n

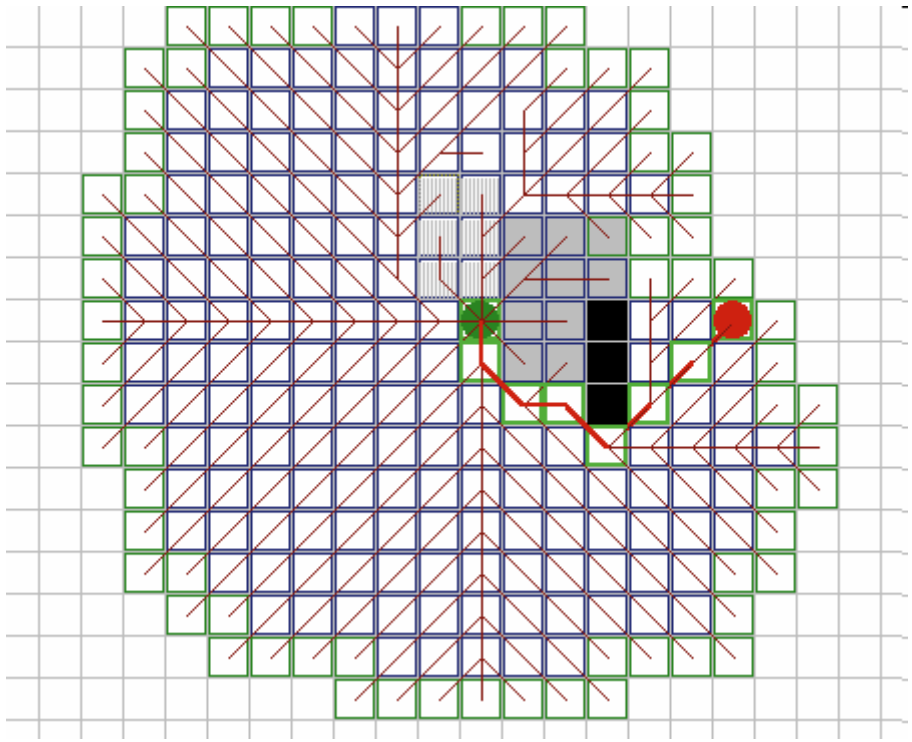
II.9.2. Алгоритм Дейкстры

- Создаем **приоритетную очередь** Open
- Значения всех **g(n)** задаем бесконечностью
- **g**(**Start**) ← **0**
- Open ← **Start** (кладем в очередь)
- **Пока** [Open не пуста]
  - Open → **N** (извлекаем из очереди)
  - **Если** [**N** = **Goal**]
    - Конструируем путь и выходим
  - **Для каждого** [**M** (соседа узла **N**)]
    - Релаксируем **M**
      - **Если** [**g**[**M**] > **g**[**N**] + **g**(**N**,**M**)]
        - **g**[**M**] ← **g**[**N**] + **g**(**N**,**M**)
    - **Parent**[**M**] ← **N**
    - **Если** [**M** не находится в Open]
      - Open ← **M**
- Путь не найден, выходим

II.9.3. Алгоритм Дейкстры: преимущества и недостатки

- Преимущества:
  - Работает во взвешенных средах
  - Обновляет узлы при нахождении лучшего пути к ним
- Недостатки:
  - Игнорирует направление к цели

II.9.4. Алгоритм Дейкстры: иллюстрация работы



II.10.1. Best-First Search

- Алгоритм «Лучший–Первый» был создан с целью исправить основной недостаток предыдущих алгоритмов: игнорирование направления к цели
- Используется эвристический поиск

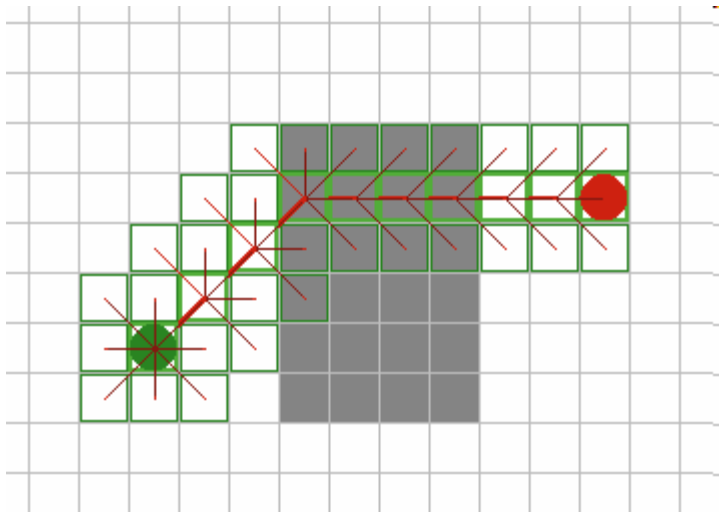
II.10.2. Best-First Search: алгоритм

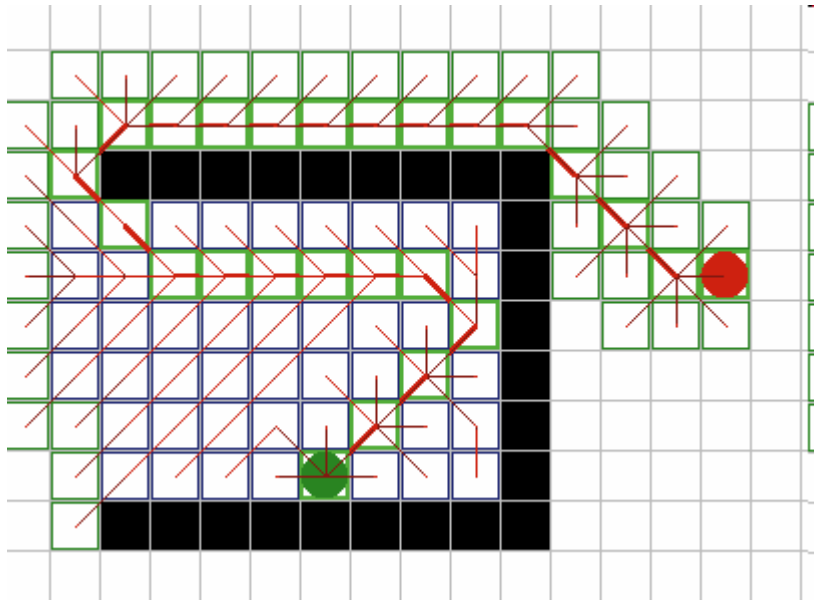
- Создаем **приоритетную очередь** Open
- Open ← **Start** (кладем в очередь)
- **Пока** [Open не пуста]
  - Open → **N** (извлекаем из очереди)
  - **Если** [**N** = **Goal**]
    - Конструируем путь и выходим
  - **Для каждого** [**M** (соседа узла **N**)]
    - **Если** [**M** не находится в Open]
      - Присваиваем M оценку с помощью некоторой эвристической функции, например, расстояния до цели
      - Open ← **M**
- Путь не найден, выходим

II.10.3. Best-First Search: преимущества и недостатки

- Преимущества:
  - Высокая скорость работы
  - Не игнорируется направление к цели
- Недостатки:
  - Не работает во взвешенных средах, не обходит зону с высокой стоимостью
  - Создает изгибающиеся, а не прямые пути вокруг препятствия

II.10.4. Best-First Search: иллюстрации недостатков





II.11.1. Алгоритм A*: введение

- Наилучший алгоритм поиска оптимальных путей
- Сочетает в себе достоинства алгоритмов Дейкстры и Best-First Search: учет длины предыдущего пути (Дейкстра) и использование эвристики (B-F Search)
- Сортирует все узлы по приближению наилучшего маршрута идущего через узел

II.11.2. Алгоритм A*: эвристическая функция

- Введем эвристическую функцию для каждого узла **n**:

  $$f(n) = g(n) + h(n)$$

  где:

  - **g**(**n**) - наименьшая стоимость прибытия в узел **n** из точки старта
  - **h**(**n**) - эвристическое приближение стоимости пути к цели от узла **n**

II.11.3. Алгоритм A*: инициализация

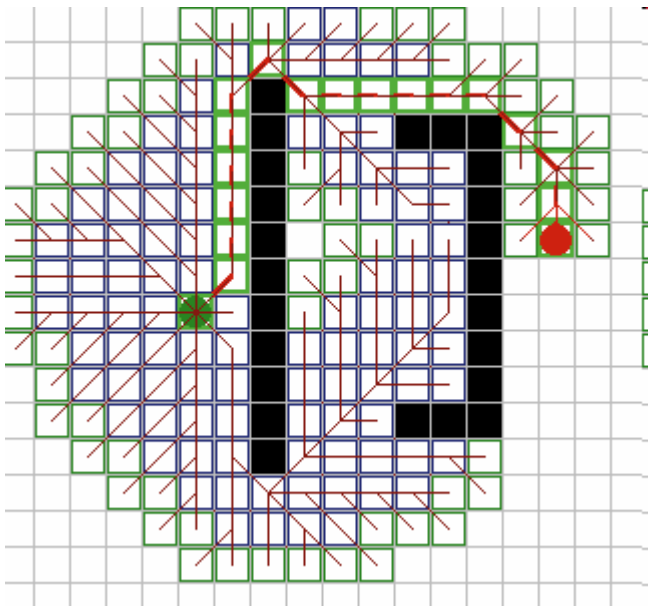
- Создаем **приоритетную очередь** Open
- Создаем **список** Closed
- Значения всех стоимостей **g(n)** задаем бесконечностью
- **g**(**Start**) ← **0**
- **h**(**Start**) ← Heuristic(**Start**) (эвристическая оценка)
- f(**Start**) ← g(**Start**) + h(**Start**)
- Open ← **Start** (кладем в очередь)

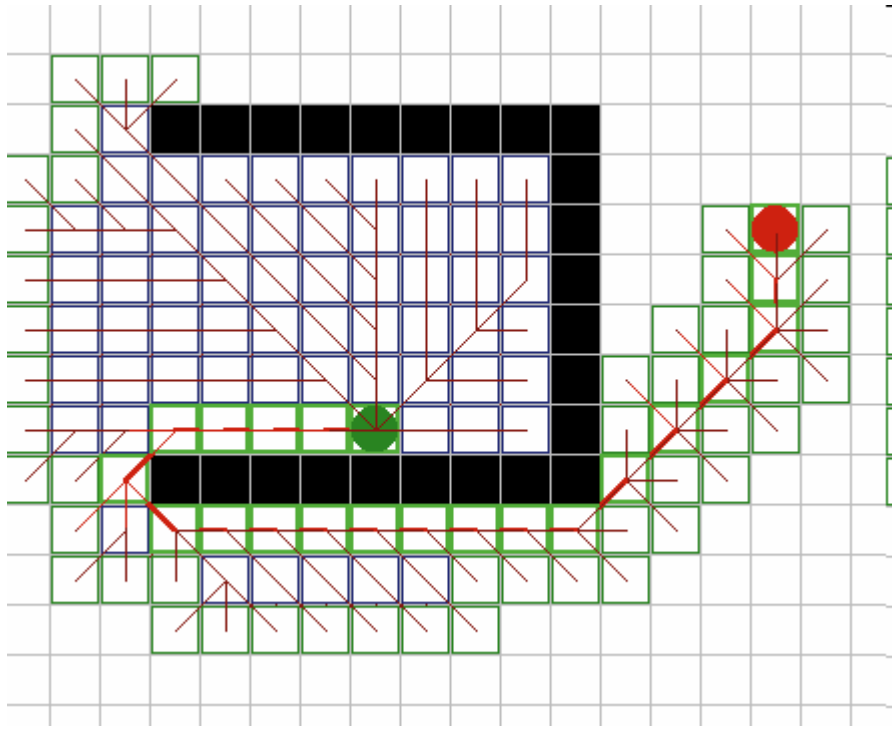
II.11.4. Алгоритм A*: основной цикл

- **Пока** [Open не пуста]
  - Open → **N** (извлекаем из очереди)
  - **Если** [**N** = **Goal**]
    - Конструируем путь и выходим
  - **Для каждого** [**M** (соседа узла **N**)]
    - **Если** [**M** находится в Open или Closed и g[**M**] ≤ g[**N**] + g(**N**,**M**)]
      - Пропускаем **M**
    - **Parent[M]** ← **N**
    - Релаксируем **M**
      - **Если** [g[**M**] > g[**N**] + g(**N**,**M**)]
        - g[**M**] ← g[**N**] + g(**N**,**M**)
    - h(**M**) ← Heuristic(**M**) (эвристическая оценка)
    - f(**M**) ← g(**M**) + h(**M**)
    - **Если** [**M** находится в Closed]
      - Closed ► **M** (удаляем из списка)
    - **Если** [M не находится в Open]
      - Open ← **M**
  - Closed ← **N**
- Путь не найден, выходим

II.11.5. Алгоритм A*: преимущества

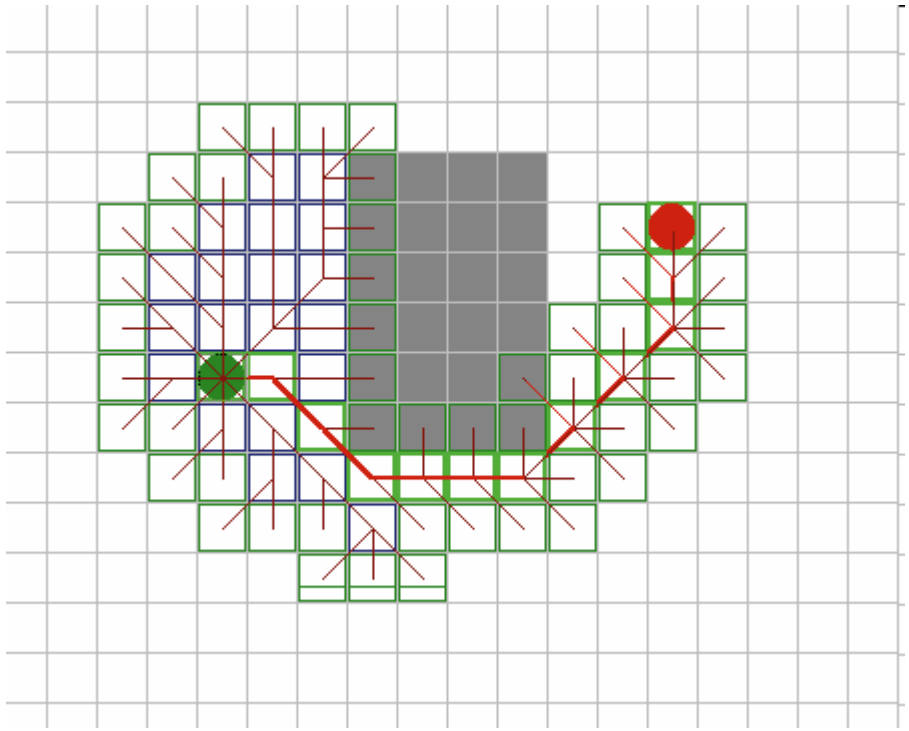
- Справляется с проблемными для других алгоритмов ситуациями





II.11.5. Алгоритм A*: преимущества

- Обход зон с высокой стоимостью прохождения

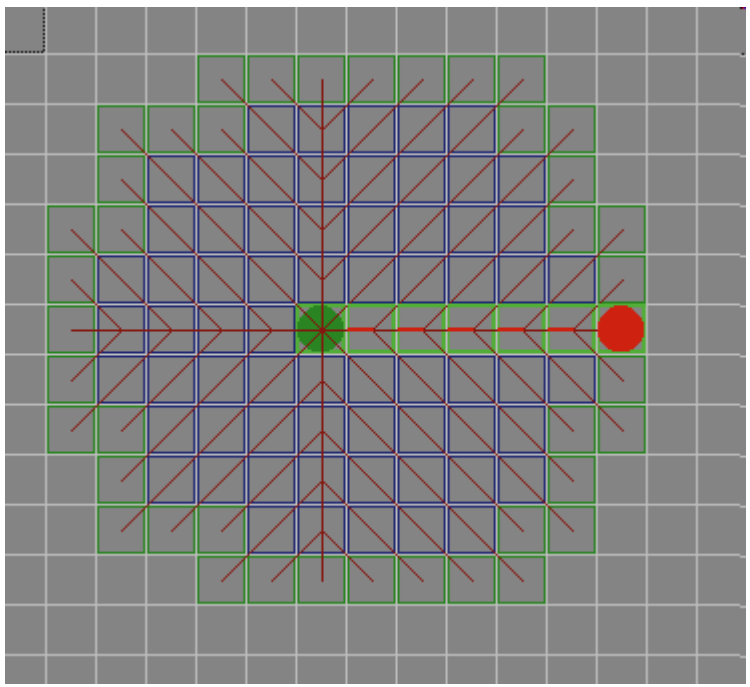


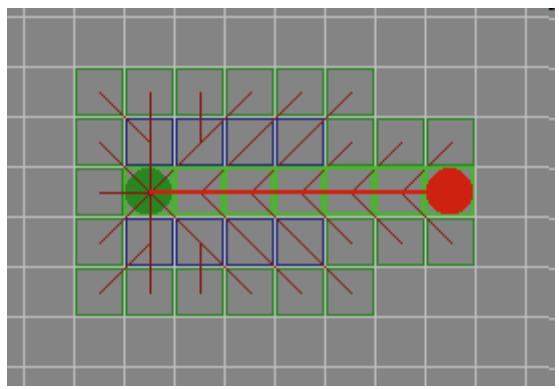
II.11.5. Алгоритм A*: преимущества

- Гибкость - при необходимости, состоянием может быть не текущая ячейка, а ориентация и скорость (например, при поиске пути для машины - их радиус поворота становится хуже при большей скорости)
- Доступность эвристики **h(n)** – в простейшем случае она является расстоянием Манхеттена до цели

II.11.6. Алгоритм A*: недостатки

- Качество работы алгоритма сильно зависит от качества эвристического приближения h(n); если приближение будет низким, то будет наблюдаться резкое ухудшение работы алгоритма





II.11.6. Алгоритм A*: недостатки

- При применении алгоритма на больших картах могут возникнуть проблемы с памятью из-за разрастания приоритетной очереди Open и списка Closed

II.11.7. Алгоритм A*: оптимизации

- **Лучевой поиск**: наложение ограничений на количество узлов в приоритетной очереди Open; когда она полна и необходимо добавить новый узел, просто выбрасывается узел с наихудшим значением; список Closed также может быть уничтожен, если каждая ячейка хранит в себе длину наилучшего пути; минус - не гарантируется оптимальность пути
- **Иерархический поиск**: разобьем карту на некоторые связные области и выберем точки на границах этих областей; т.о. задача разбивается на несколько более мелких (ищем путь до точек на границе областей)

II.11.7. Алгоритм A*: оптимизации

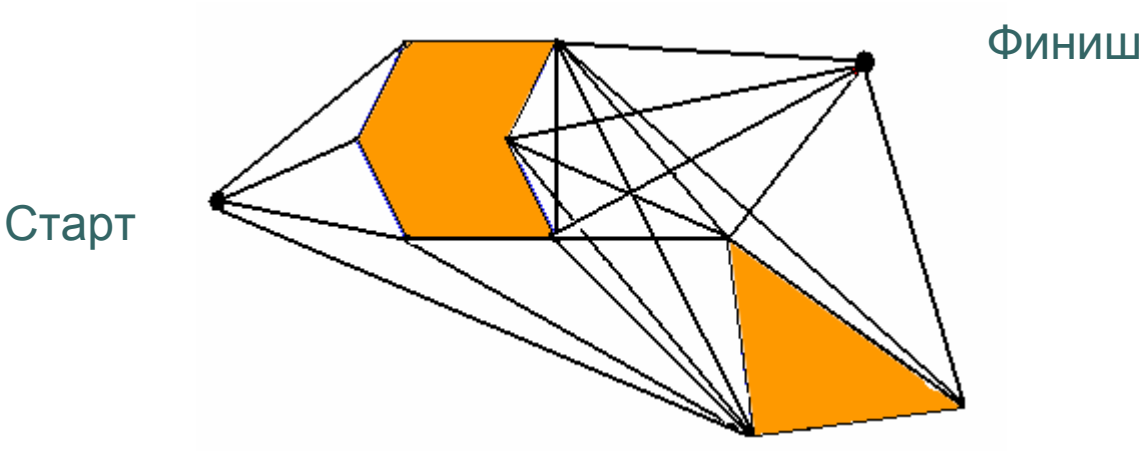
- Алгоритм последовательных приближений (IDA*): избавляет от необходимости хранить списки Open и Closed:
  - Делаем простой рекурсивный поиск, собираем наколенную стоимость пути **g(n)**
  - Прекращаем поиск при достижении заданного значения **f(n)** = **g(n)** + **h(n)**
  - В начале возьмем предел глубины поиска равным **h(Start)**
  - Если на текущей итерации путь не найден, установим новый предел глубины поиска, равным минимальному значению **f(n)**, которое превысило прежнюю границу

III. Трассировка в известных непрерывных средах

III.1. Графы видимости

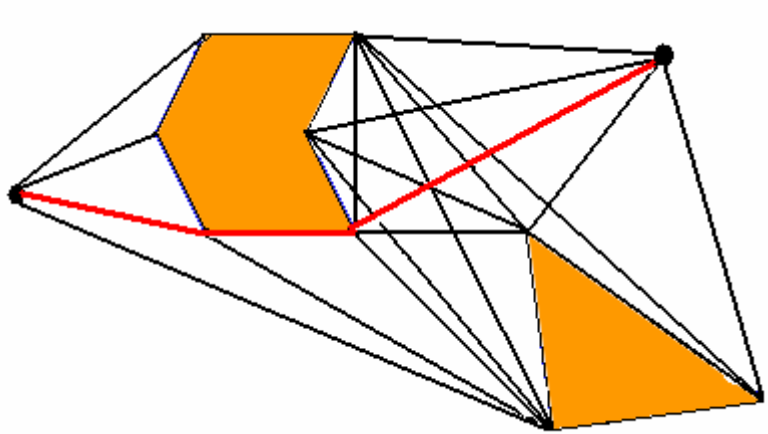
III.1.1. Суть метода

- В этом методе строится неориентированный граф между начальной, конечной точками и всеми вершинами препятствий окружающего пространства



III.1.2. Основное свойство

- Основное свойство:
  - Строим ребро между любыми двумя вершинами, если оно не проходят через препятствие



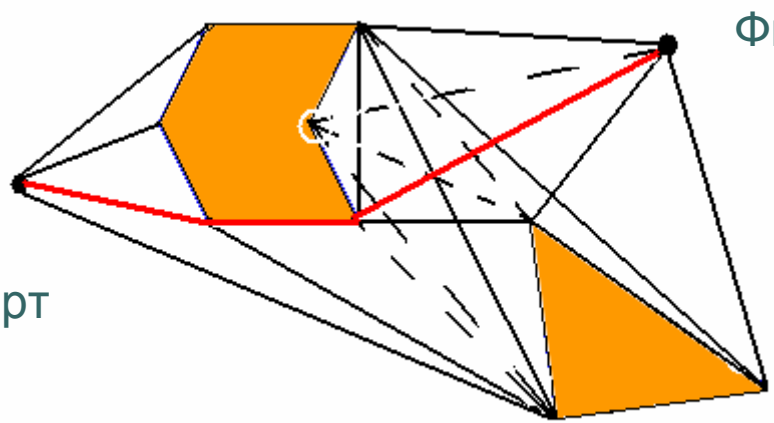
III.1.3. Некоторые соображения

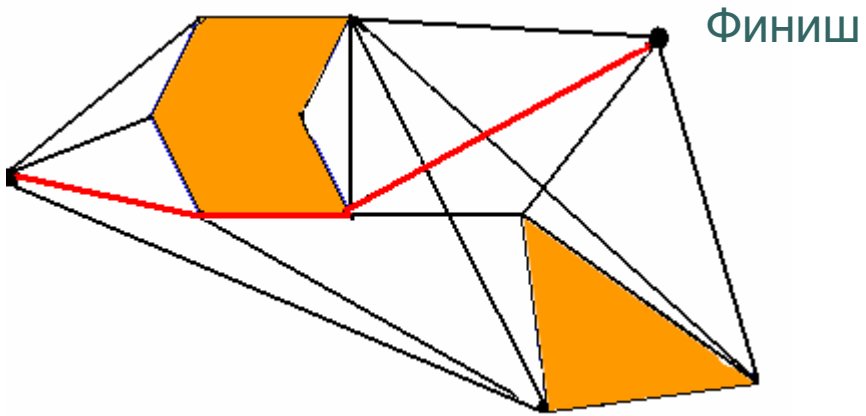
- Для нахождения пути используем алгоритмы на графах, например, алгоритм Дейкстры
- Очевидно, что в случае нахождения препятствия на пути робота путь будет проходить по границе данного препятствия

III.1.4. Упрощение графа видимости

- Очевидно, что некоторые вершины в процессе нахождения пути не будут играть роли
- Упрощение графа видимости может быть совершено путем исключения таких вершин

III.1.4. Упрощение графа видимости (пример)





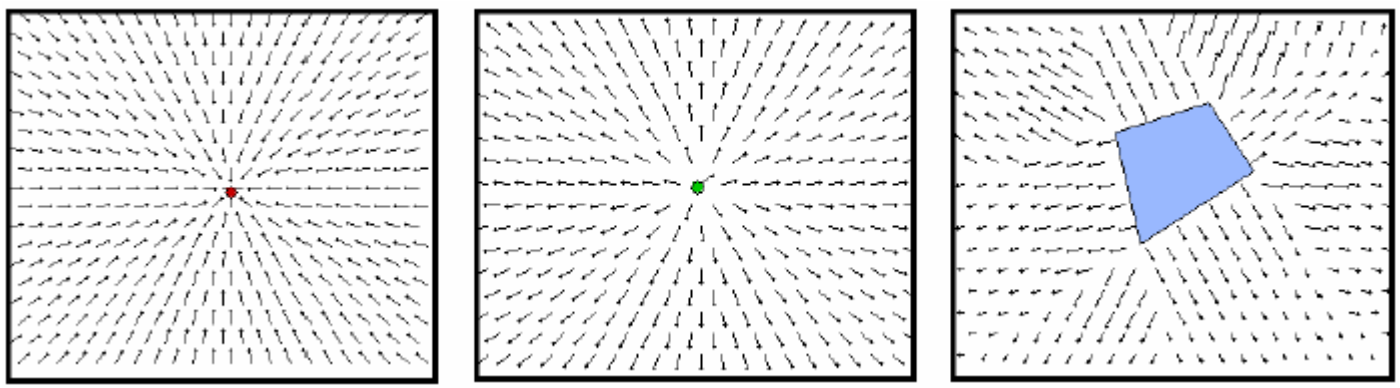
III.2. Метод потенциальных полей

III.2.1. Основные идеи

- Навигация методом потенциальных полей базируется на идее того, что окружающие объекты притягивают или отталкивают робота в процессе движения
- Робот вычисляет вектор, который является функцией целевой точки и окружающий препятствий
- Робот движется по направлению вектора, пока не достигнет цели назначения, через определенные промежутки времени рассчитывая вектор

III.2.2. Основные понятия

- Основные правила:
  - Робот притягивается к конечной точке
  - Отталкивается от начала движения
  - Отталкивается от окружающих препятствий



III.2.2. Основные понятия

- Каждый конкретный вектор определяет направление вектора движения робота, в конкретной точке
  - Очевидно, что на каждом шаге роботом рассчитывается только ОДИН вектор

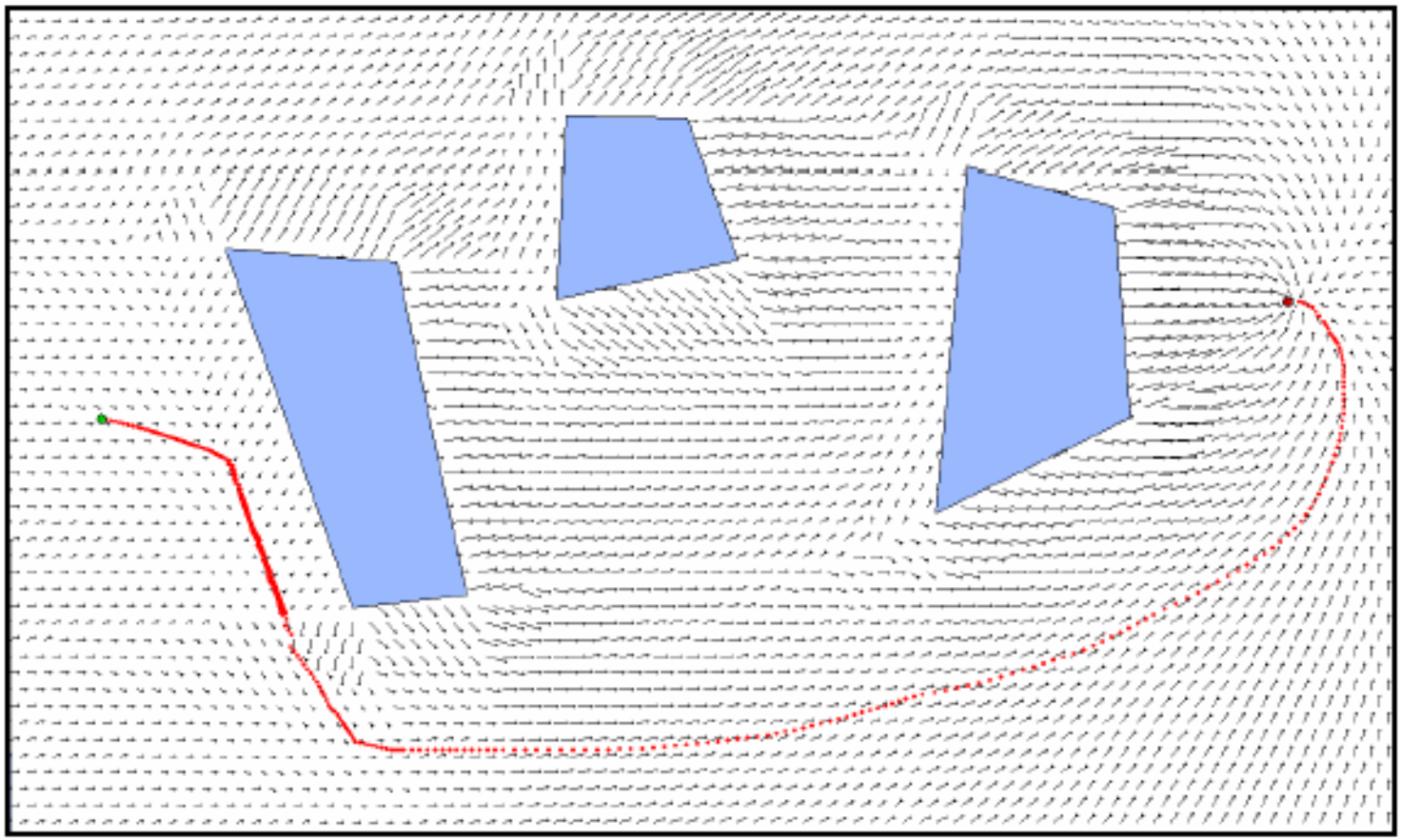
III.2.3. Magnitude

- У потенциального поля есть параметр Magnitude, который показывает “необходимость” передвижения робота в указанном направлении
  - Например: вблизи препятствия такой параметр будет значительно больше, чем вдали от этого же препятствия

III.2.4. Суперпозиция потенциальных полей

- Потенциальное поле может быть “составлено” из нескольких отдельных полей, которые обеспечивают роботу перемещение к точке назначения
  - Поля начальной, конечной точек
  - Поля препятствий
- Робот рассчитывает новый вектор в конкретной точке, исходя из полей, которые влияют на движение в этой точке

III.2.4. Суперпозиция потенциальных полей



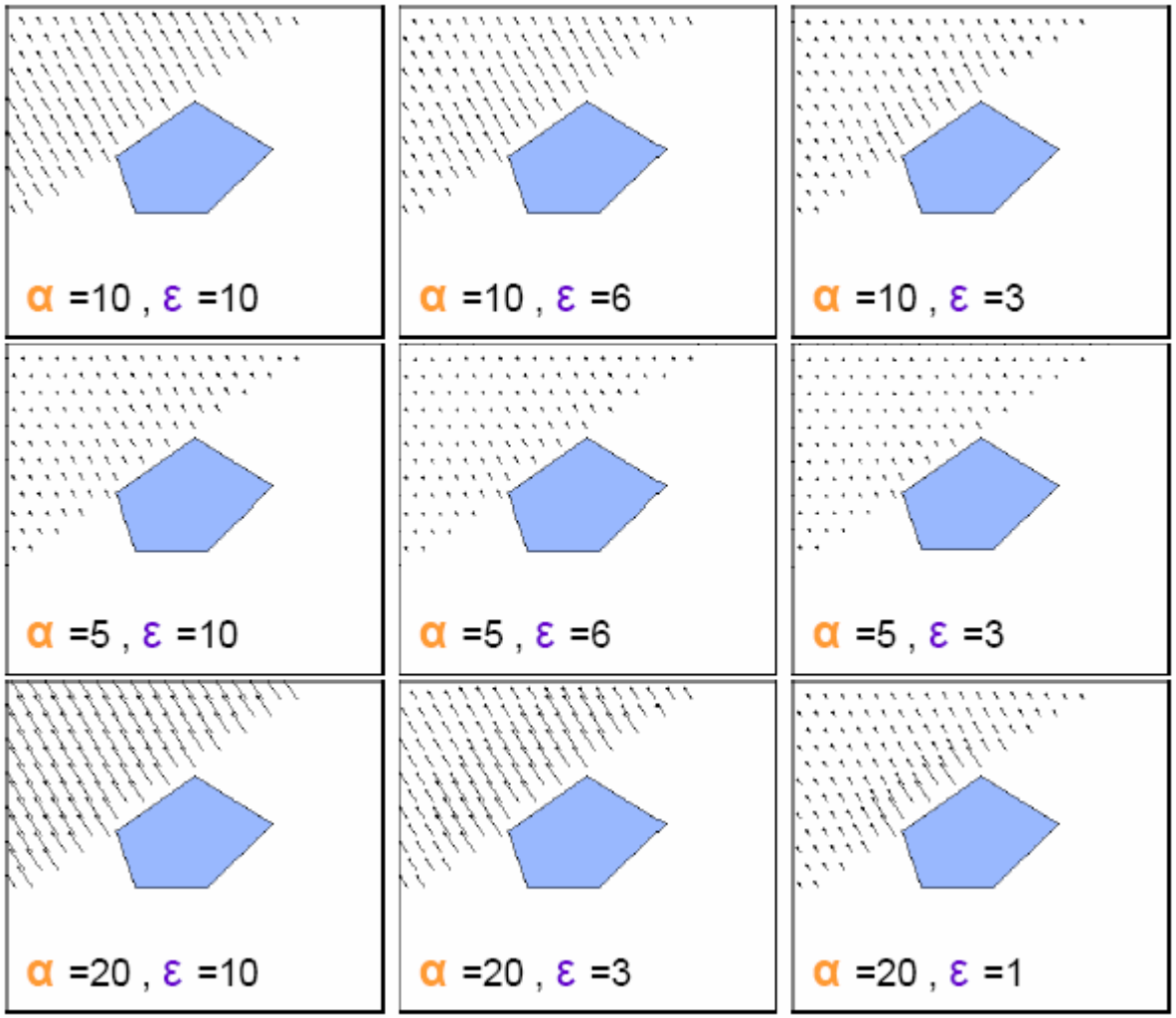
III.2.5. Расчет потенциального поля

- Расчет значения вектора движения для определенной точки (x,y):
  - От стартовой точки (g_x,g_y)
    - Magnitude = $\alpha_g * 1 / \text{distance}((x,y) \rightarrow (g_x,g_y))$
    - Направление = угол между $((x,y) \rightarrow (g_x,g_y))$
  - Вектор от ребра преграды $(s_x,s_y) \rightarrow (d_x,d_y)$
    - Magnitude = $\alpha_{obst} / (d / (\varepsilon * \alpha_{obst}) + 1)$
    - Направление: перпендикулярно ребру

III.2.5. Расчет потенциального поля

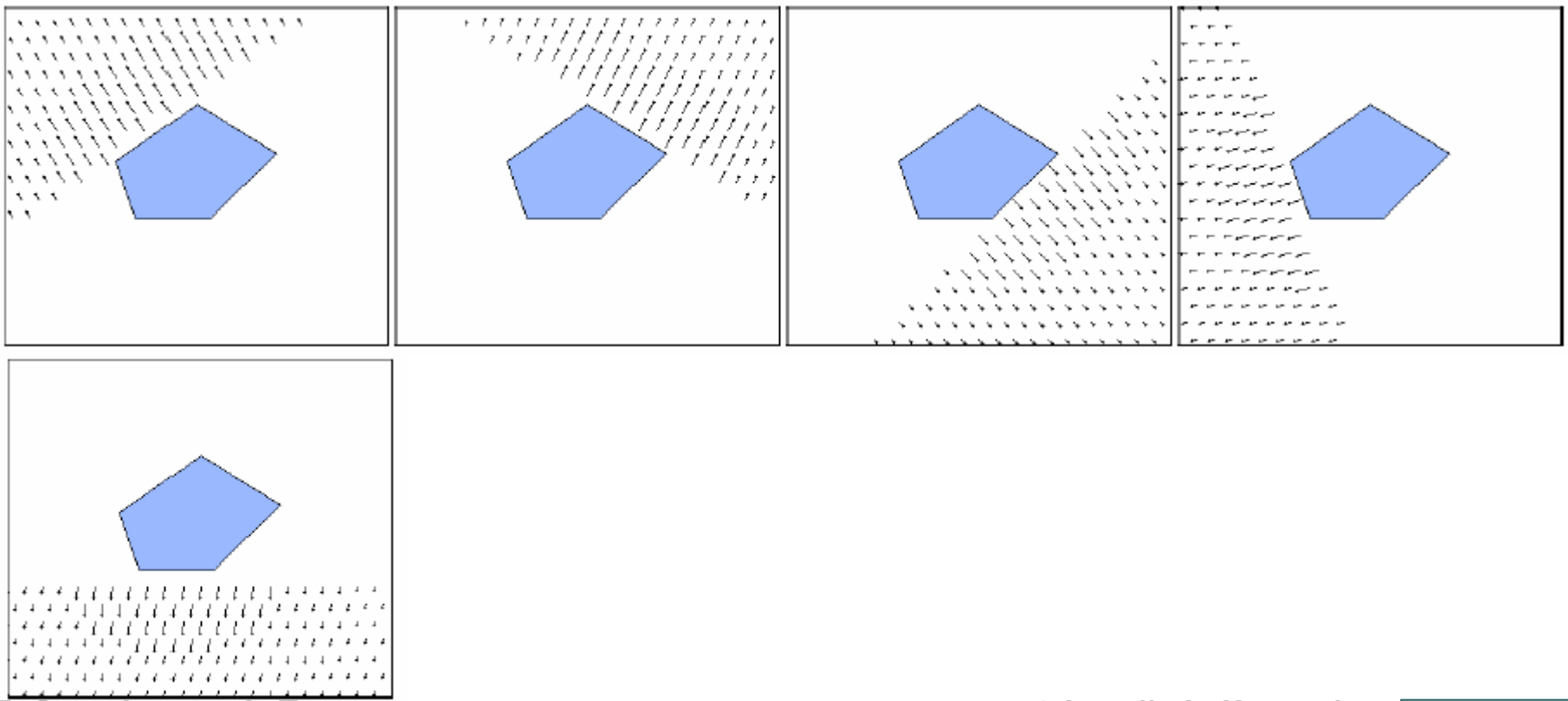
- Возможно изменение значения переменных α и ε для получения различной силы потенциального поля
  - α определяет максимальное значение Magnitude
  - ε определяет скорость изменения значения Magnitude

III.2.5. Расчет потенциального поля (пример 1)

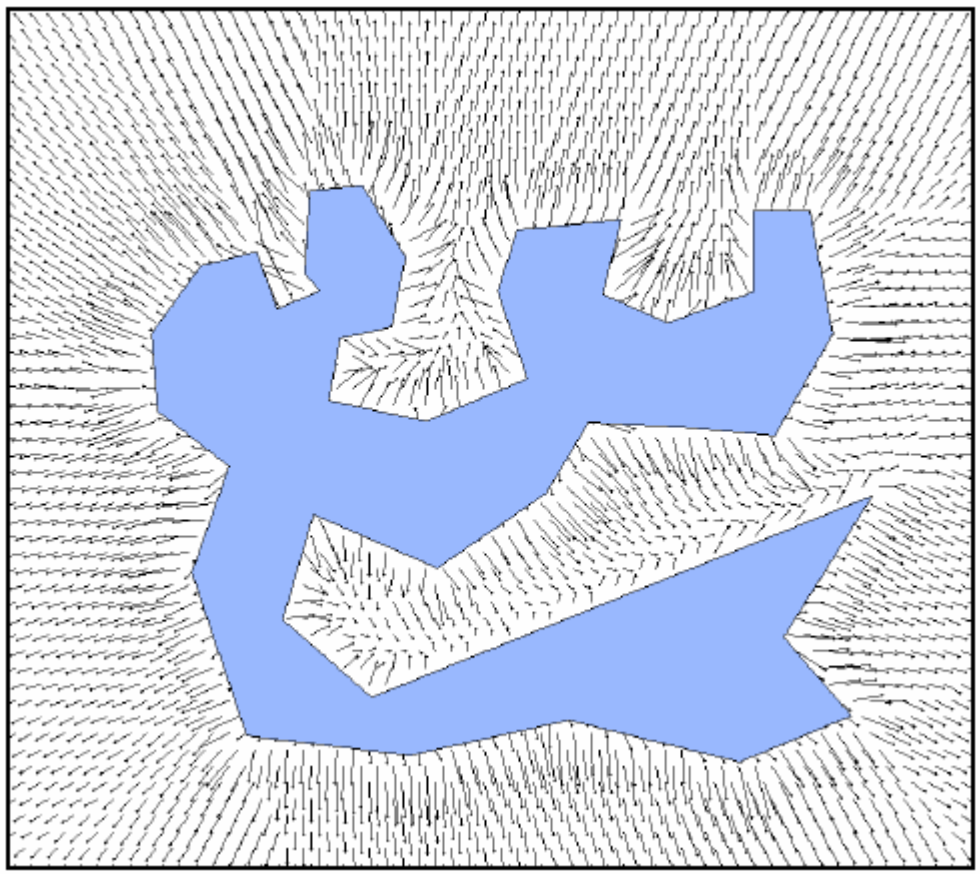


III.2.5. Расчет потенциального поля (пример 2)

- Каждое ребро производит свой вектор.

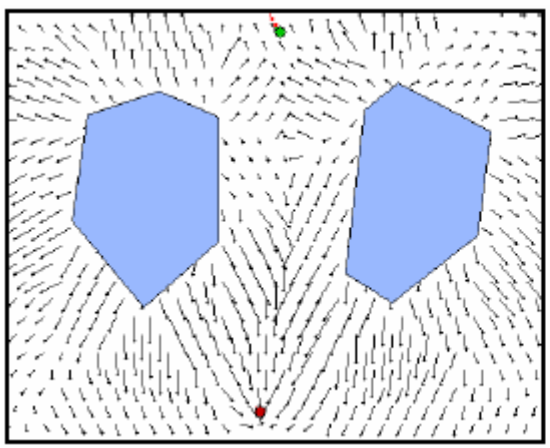


III.2.5. Расчет потенциального поля (пример 3)

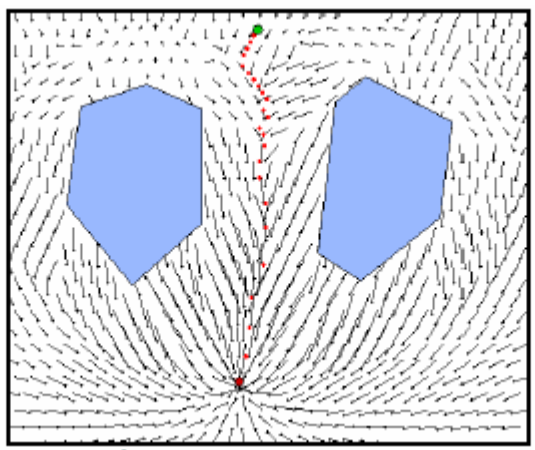


III.2.6. Проблемы поиска пути

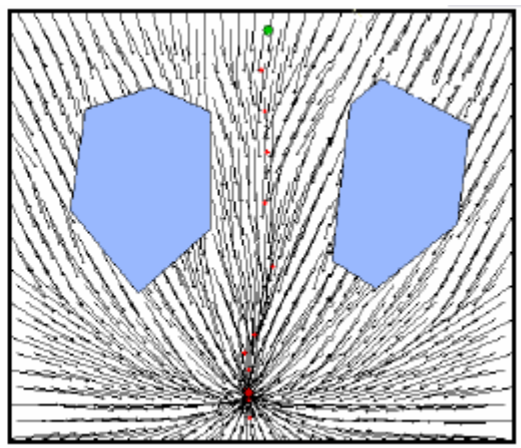
- Сила потенциального поля
  - Иногда поля, создаваемые преградами, отталкивают робота так, что препятствует нахождению решения



Слабое поле



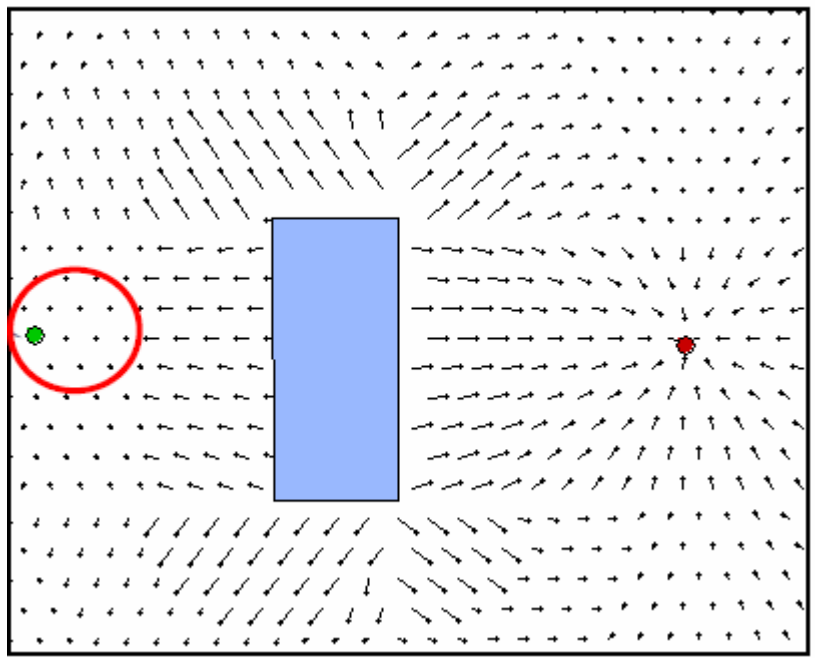
Среднее поле



Сильное поле

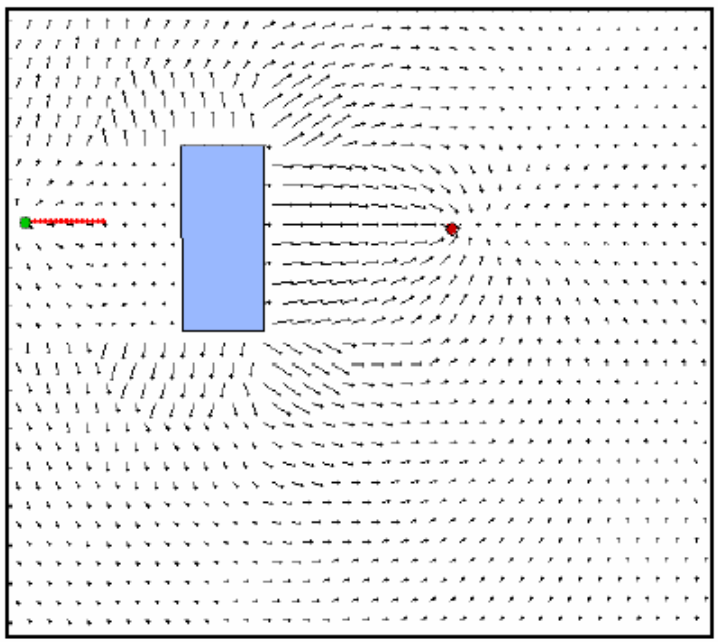
III.2.7. Локальный минимум

- В некоторых случаях могут появиться области с локальным минимумом, где робот прекратит свое движение

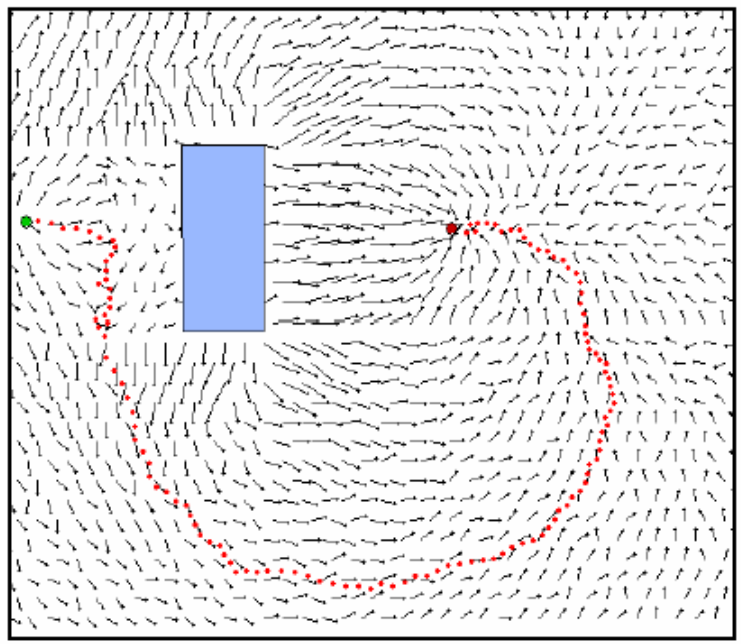


III.2.7. Локальный минимум

- Для устранения проблемы локальных минимумов применяется “шум”, но это не гарантирует правильное решение



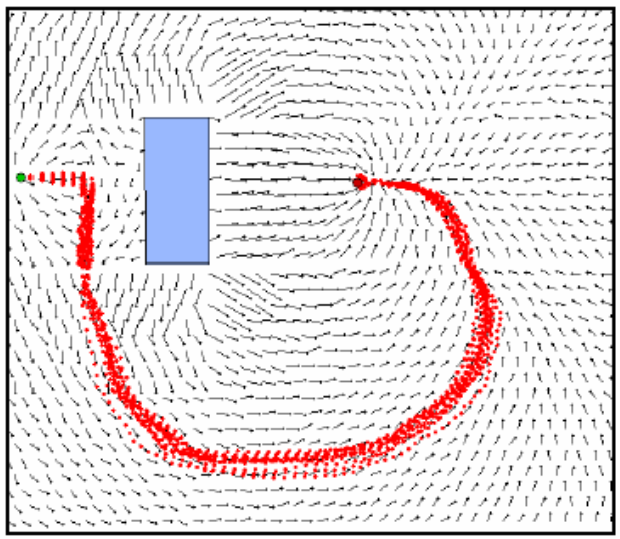
Поле без шума



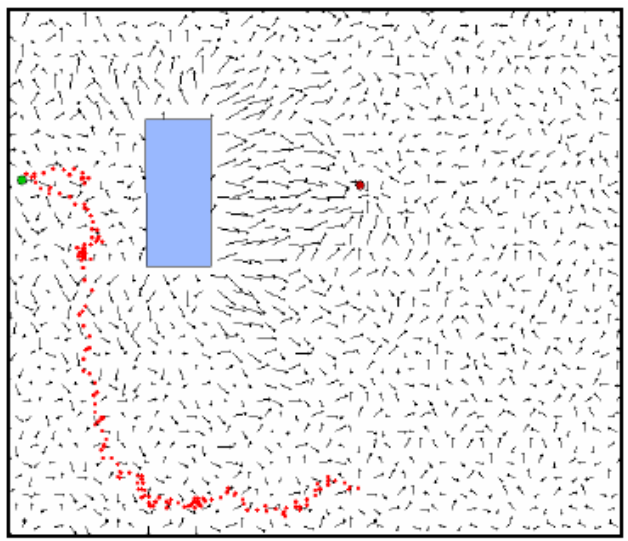
Поле с шумом

III.2.7. Локальный минимум

- Путь может варьироваться в зависимости от значений шума (случайных)
- При слишком большом шуме может возникнуть потеря решений



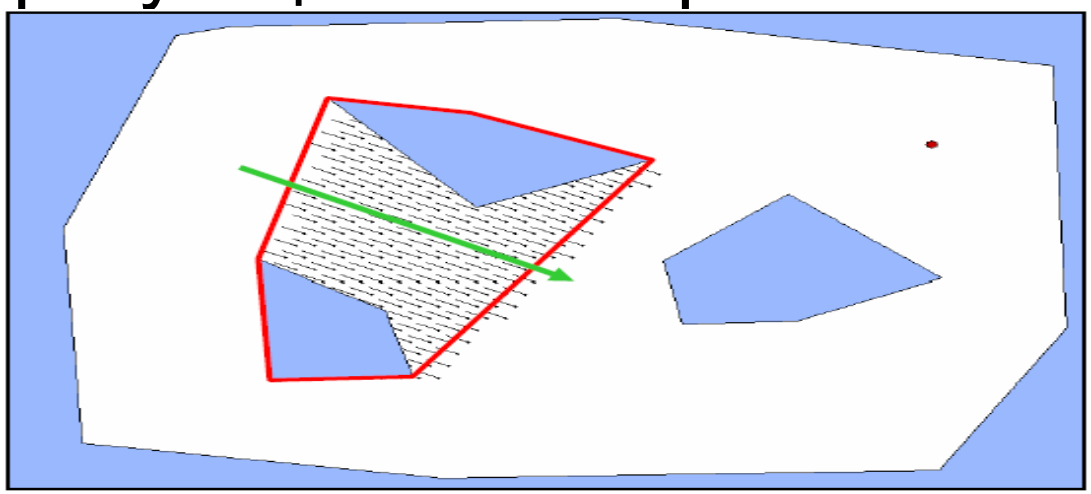
Множественный проход



Большое значение шума

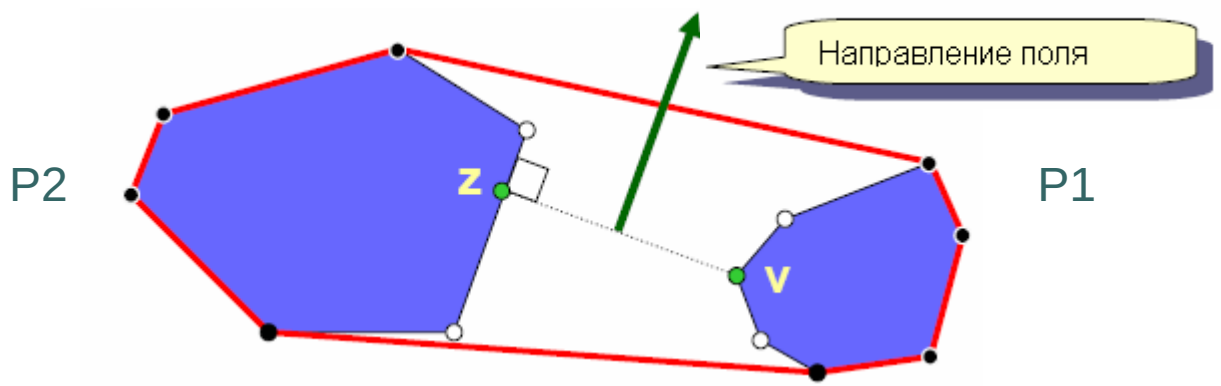
III.2.8. Коридорные поля

- Для продвижения робота между препятствиями используются “коридорные поля”, направленные в сторону точки назначения, образующие своеобразный поток

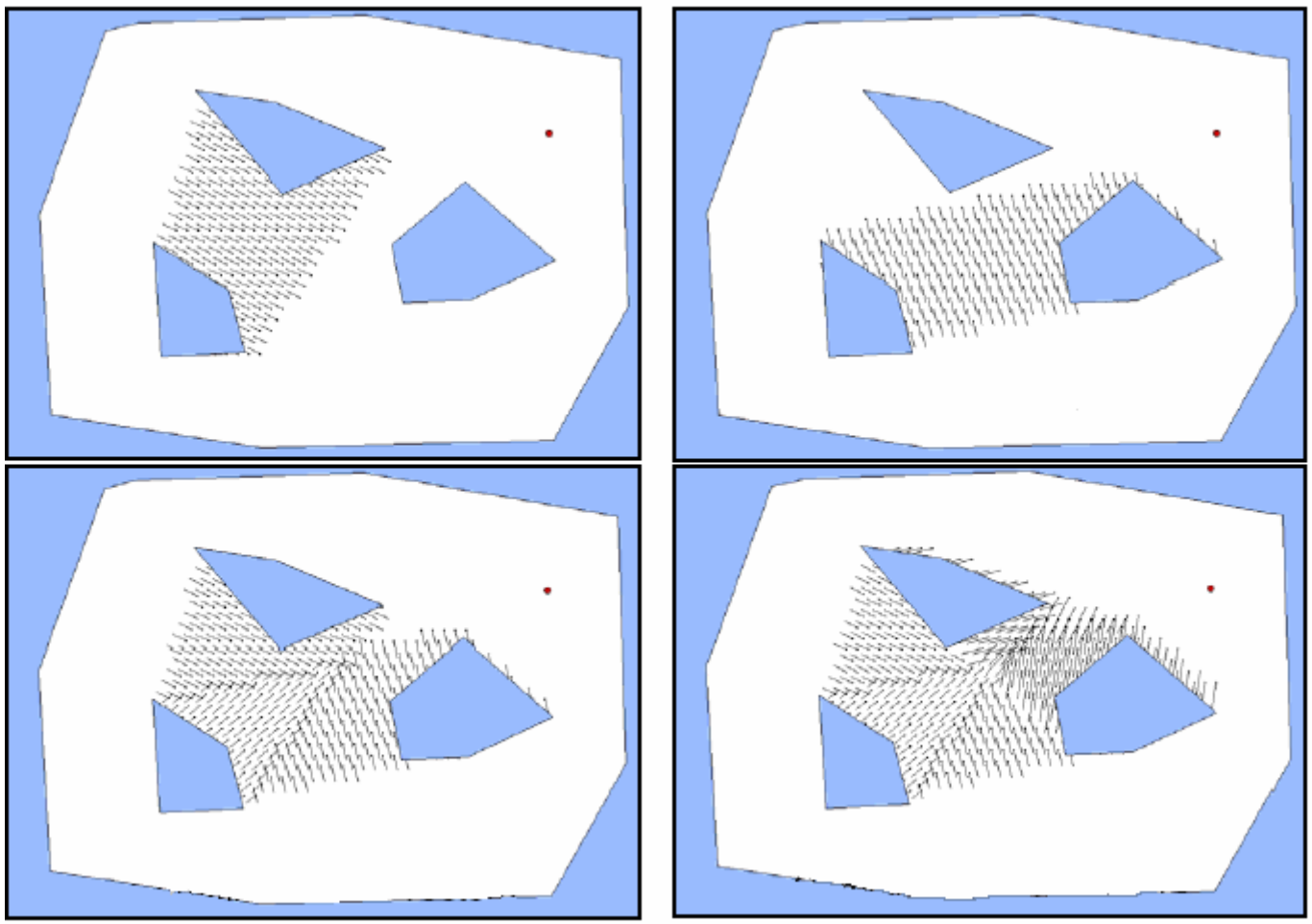


III.2.8. Расчет коридорного поля

- Для расчета коридорного поля:
  - P1 и P2 – полигоны препятствий
  - Вершина V – ближайшая к полигону P2, из нее опускается перпендикуляр на полигон P2, который определяет направление коридорного поля

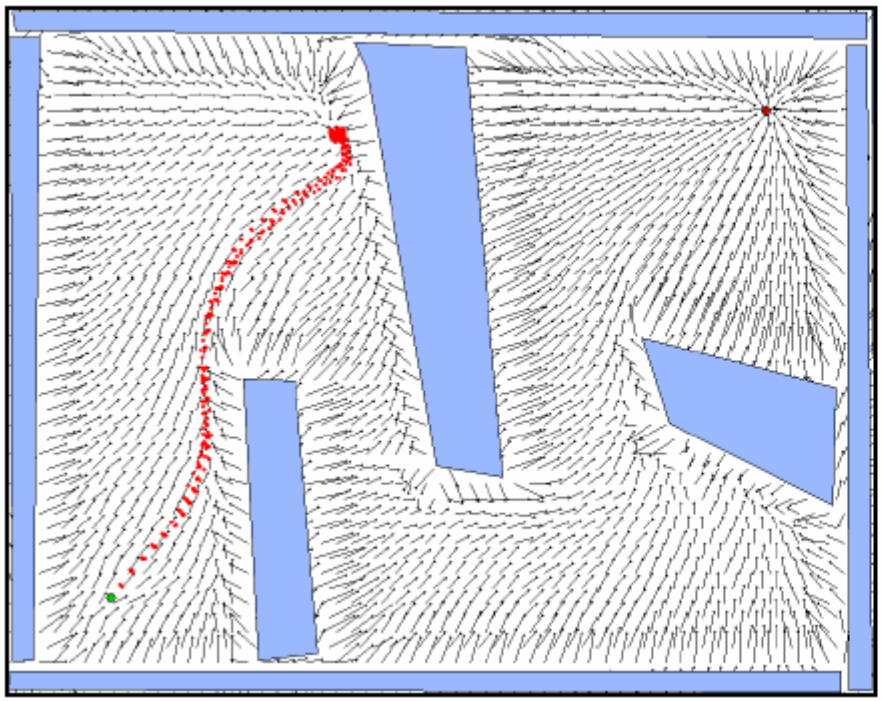


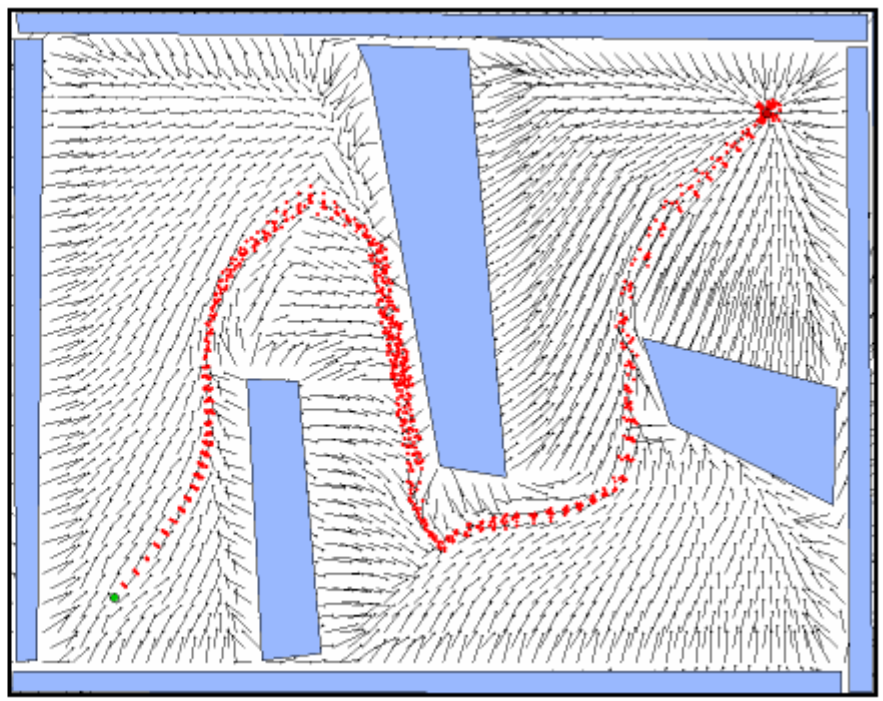
III.2.8. Коридорные поля (пример)



III.2.8. Коридорные поля

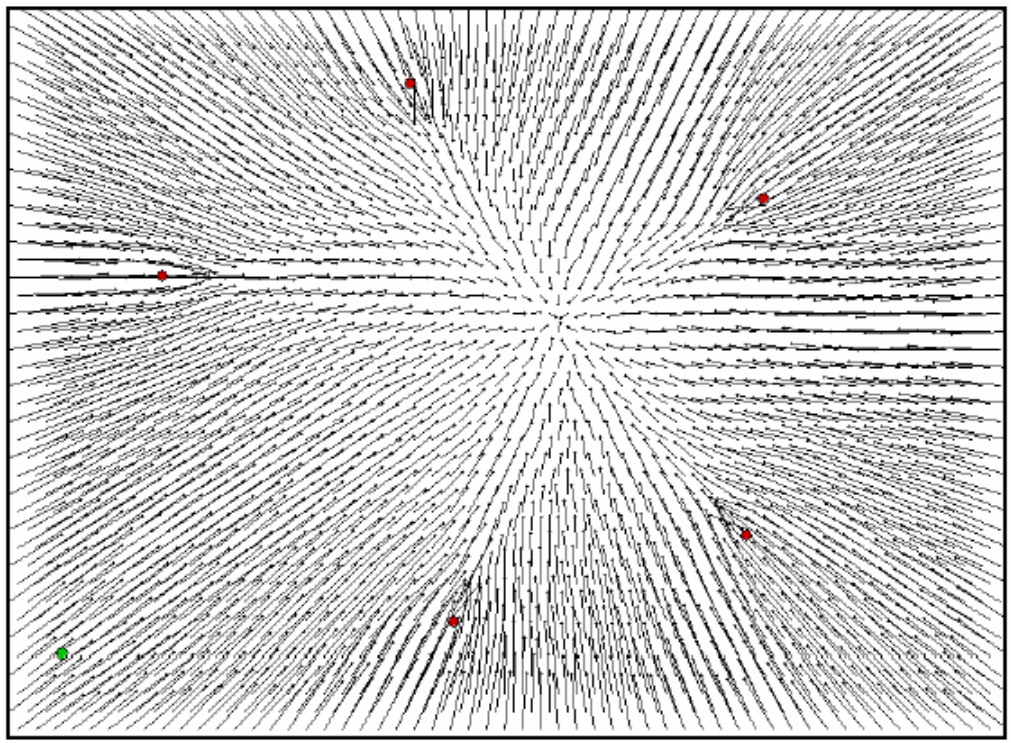
- В некоторых случаях коридорные поля позволяют роботу достичь точки назначения





III.2.9. Проблемы

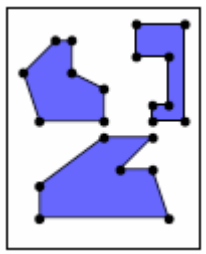
- Метод потенциальных полей не может находить путь, когда есть несколько точек назначения



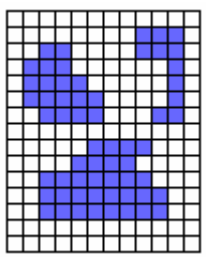
III.3. Дискретизация

III.3.1. Представления

- Представление:
  - Векторное



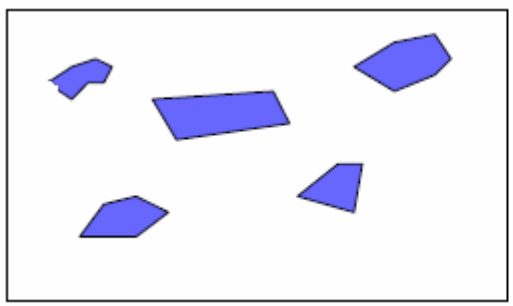
    - Хранится совокупность отрезков и полигонов
    - Обычно вектора представляют собой границы препятствий
  - Растровое

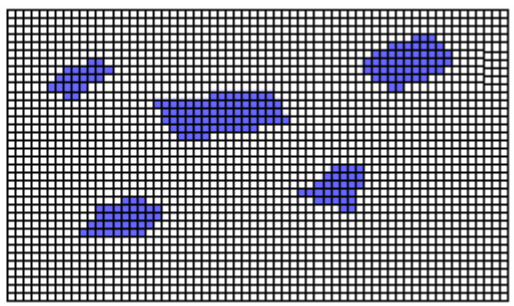


    - Карта хранится в виде сетки
    - Каждая ячейка хранит вероятность занятости

III.3.2. Выбор представления

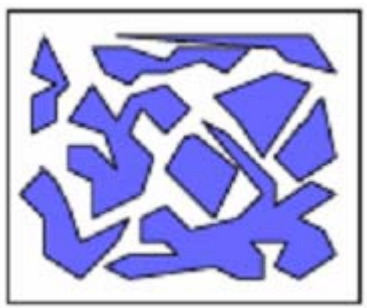
- При малом количестве простых препятствий меньше памяти используется при векторном представлении

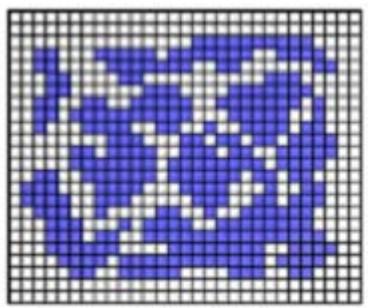




III.3.2. Выбор представления

- При большом количестве препятствий, состоящих из множества ребер, меньше памяти используется при растровом представлении



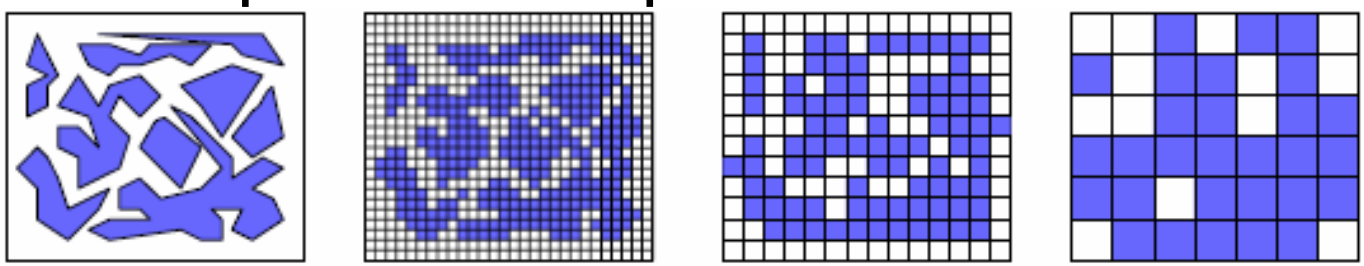


III.3.3. Свойства представлений

- Векторное представление
  - m препятствий с n вершинами
  - Память: (m*n)*2 + (m*n) = O(mn)
- Растровое представление
  - Сетка M x N требует O(MN) памяти
- Если m << M, n << N, то предпочтительнее использовать векторное представление

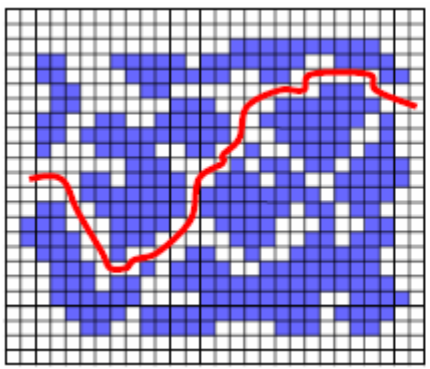
III.3.4. Растровое представление

- Точность представления зависит от разрешения растровой сетки (M x N)
- С уменьшением разрешения:
  - Уменьшается память для хранения
  - Теряется информации об окружающей среде, возможны неправильные решения

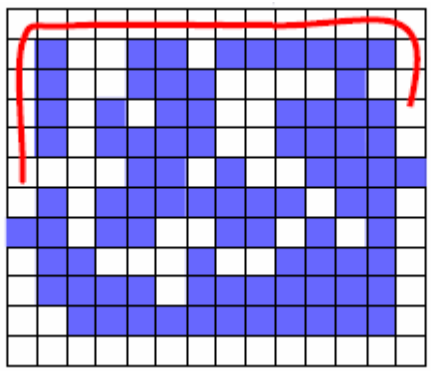


III.3.4. Растровое представление

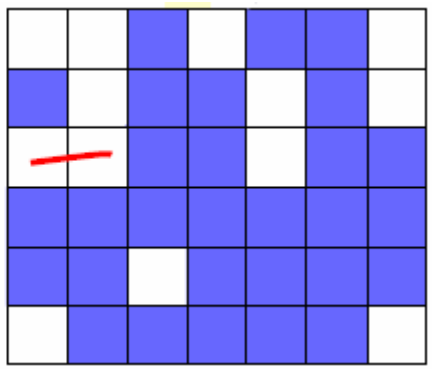
- С уменьшением разрешения возможна потеря некоторых или всех решений



Верное решение



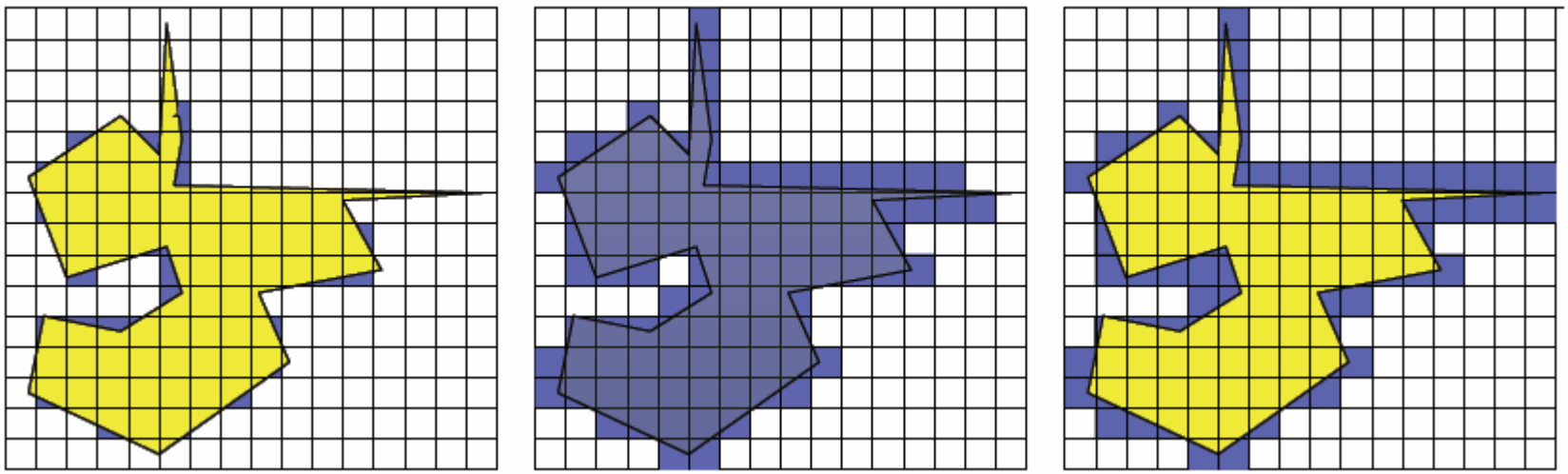
Неверное решение



Нет решений

III.3.4. Растровое представление

- Другая проблема при использовании растрового представления - это обеспечение безопасности робота при движении

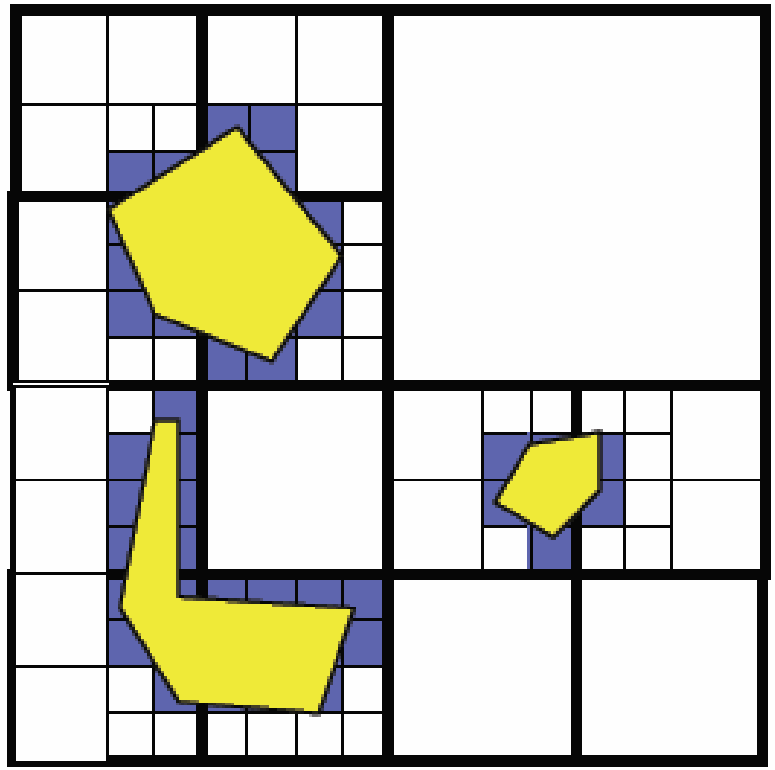


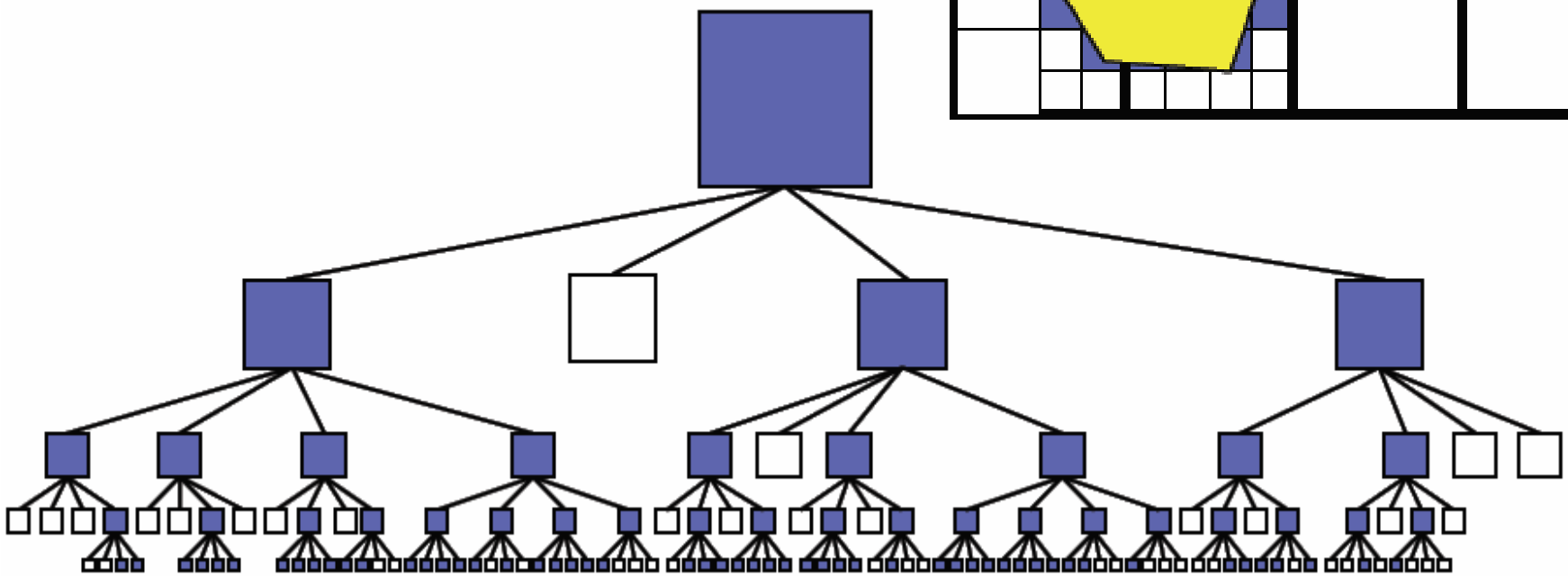
III.3.5. Хранение растрового представления

- Очевидно, хранение растра при помощи массивов неэффективно, лучше его хранить при помощи квадро-деревьев
  - Уменьшается памяти для хранения карты
  - Требуется использование более сложных алгоритмов в связи с разными размерами ячеек и необходимостью распознавания границ препятствий

III.3.5. Пример

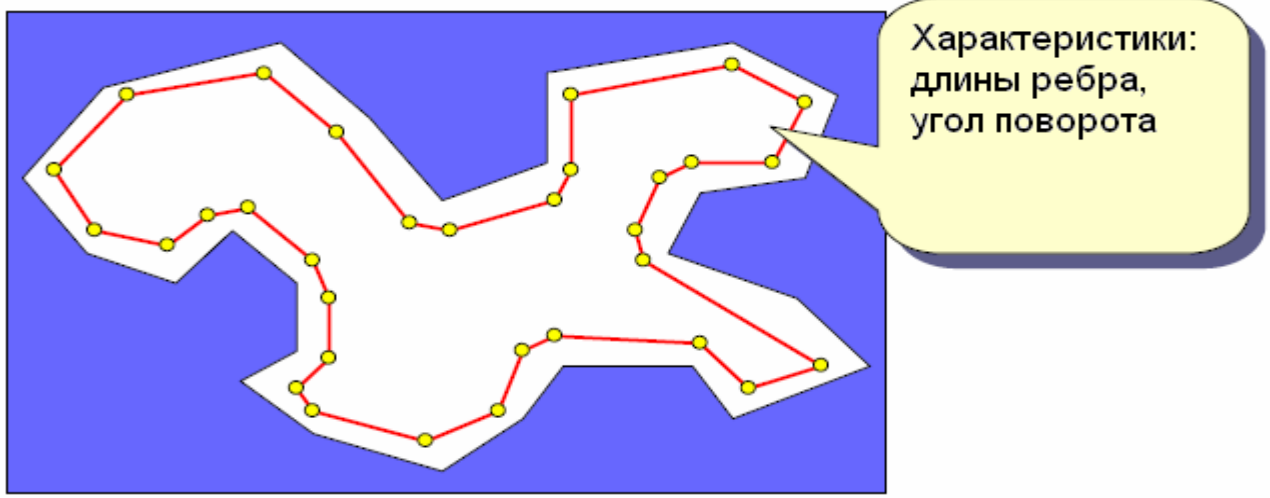
- Хранение растровой карты





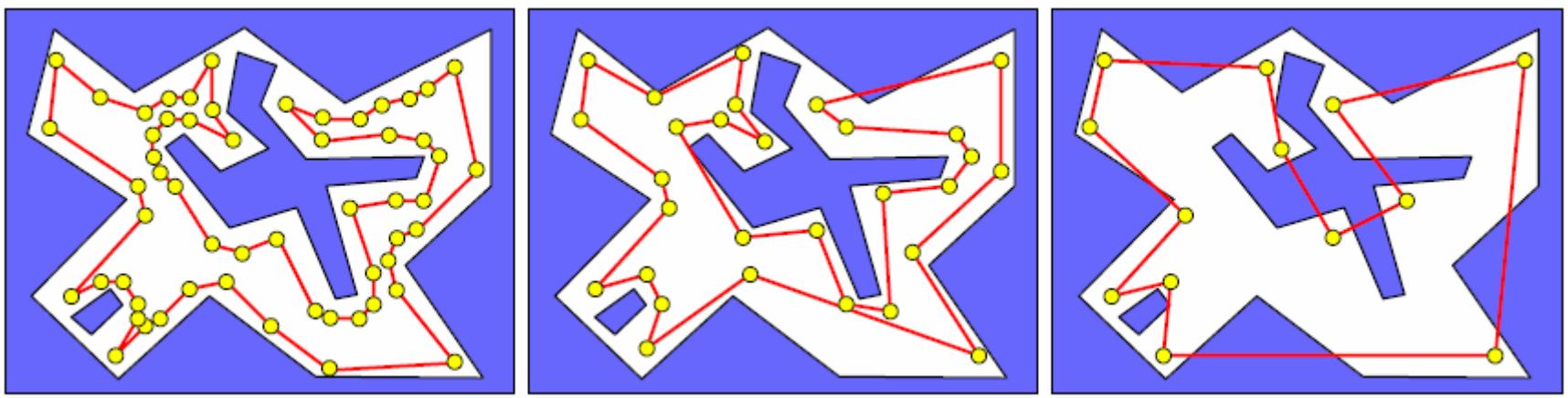
III.3.6.1. Характеристические карты (Feature Maps)

- Хранятся “характерные” черты (характеристики) окружающего пространства
  - Такими чертами могут быть: пустоты, преграды и т.п.



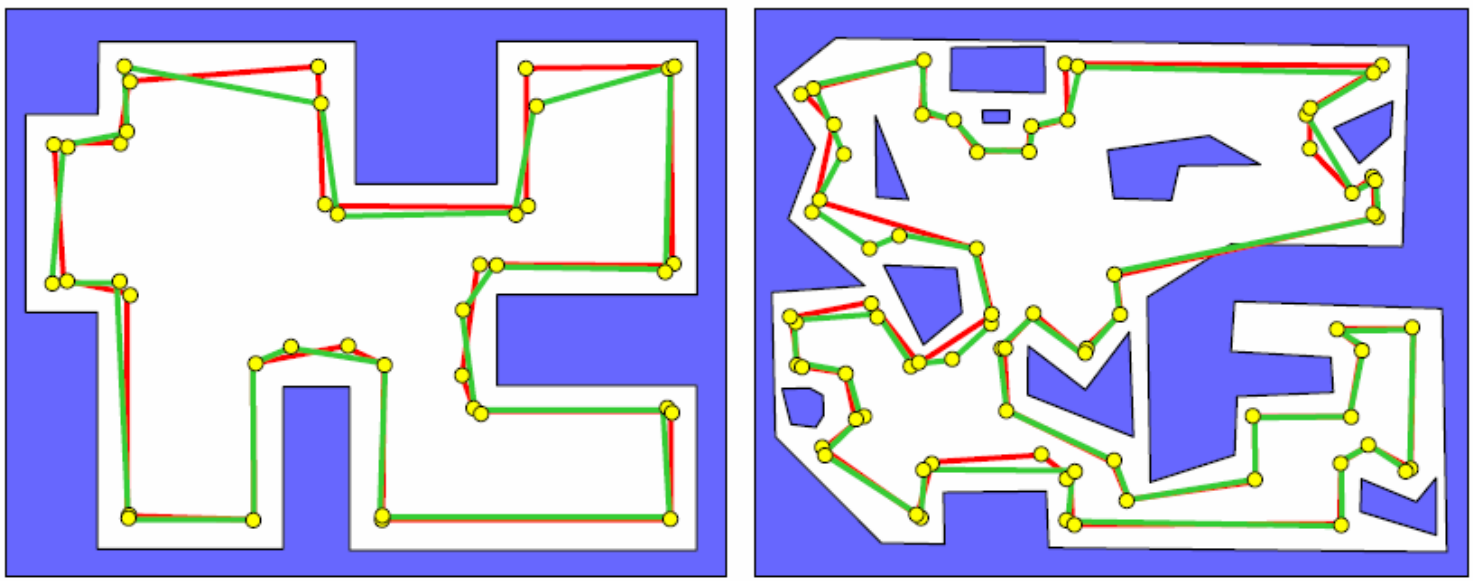
III.3.6.2. Характеристические карты (Feature Maps)

- Произведенная карта зависит от точности робота, а также от других параметров, например, минимального угла, который считается вершиной



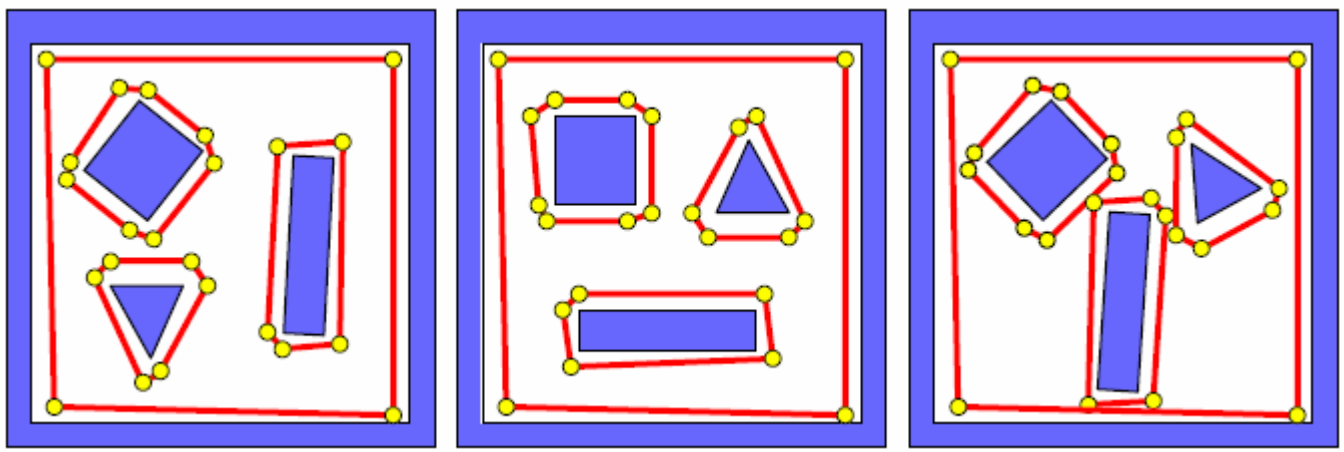
III.3.6.3. Характеристические карты (Feature Maps)

- Различные пути прохода одних и тех же препятствий, например, обход по или против часовой стрелки, производят разные карты



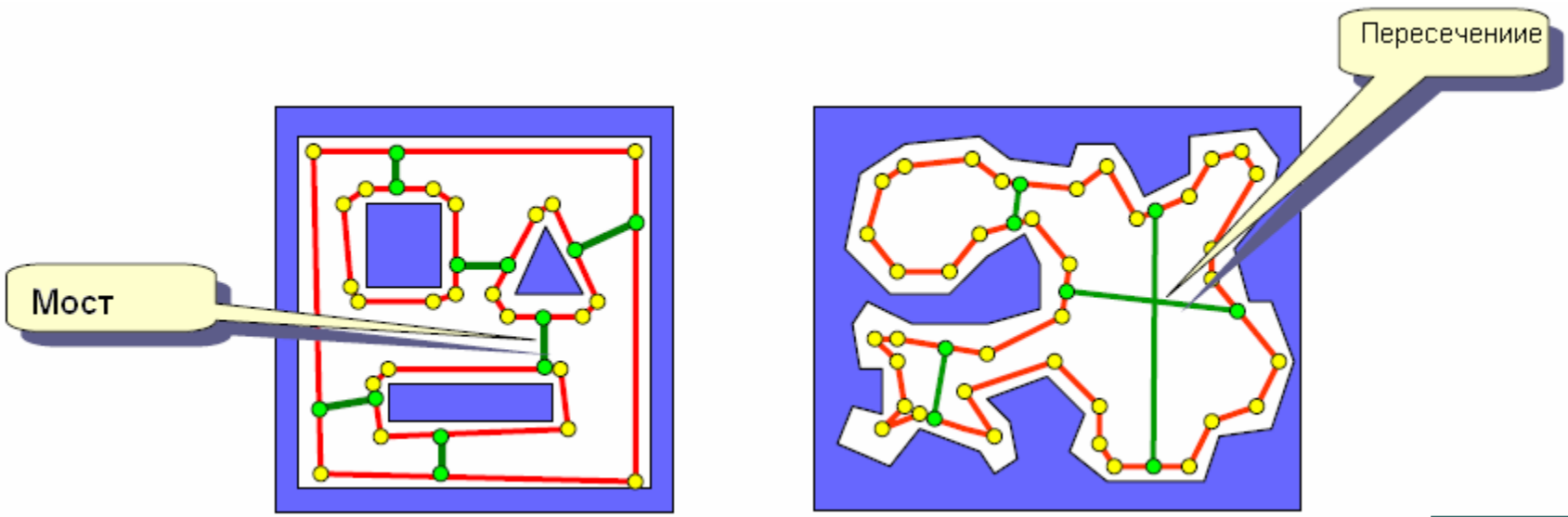
III.3.6.4. Проблемы

- При простом обходе не могут быть определены ни глобальная ориентация, ни относительная ориентация и позиция
  - Эти карты неразличимы



III.3.6.5. Мосты и пересечения

- Мосты и пересечения могут быть между различными препятствиями для создания топологического порядка для приблизительного расчета расстояния между препятствиями



III.3.6.6. Способ реализации

- Одним из способов реализации является проведение из ребра перпендикуляра:
  - Создание связи между ребром, из которого мы вышли, и ребром, с которым произошло пересечение
  - Запоминание расстояния между ними

IV. Трассировка в неизвестных средах

IV.1. Основные условия

Необходимые условия:

- Робот должен «чувствовать» препятствия и уметь двигаться вдоль них
- Цель должна быть известна, но расположение препятствий – нет
- Робот должен знать свои координаты и координаты цели

IV.2. Основные алгоритмы

- Мы рассмотрим семейство Bug, включающее в себя следующие алгоритмы:
  - Bug1
  - Bug2
  - Tangent Bug

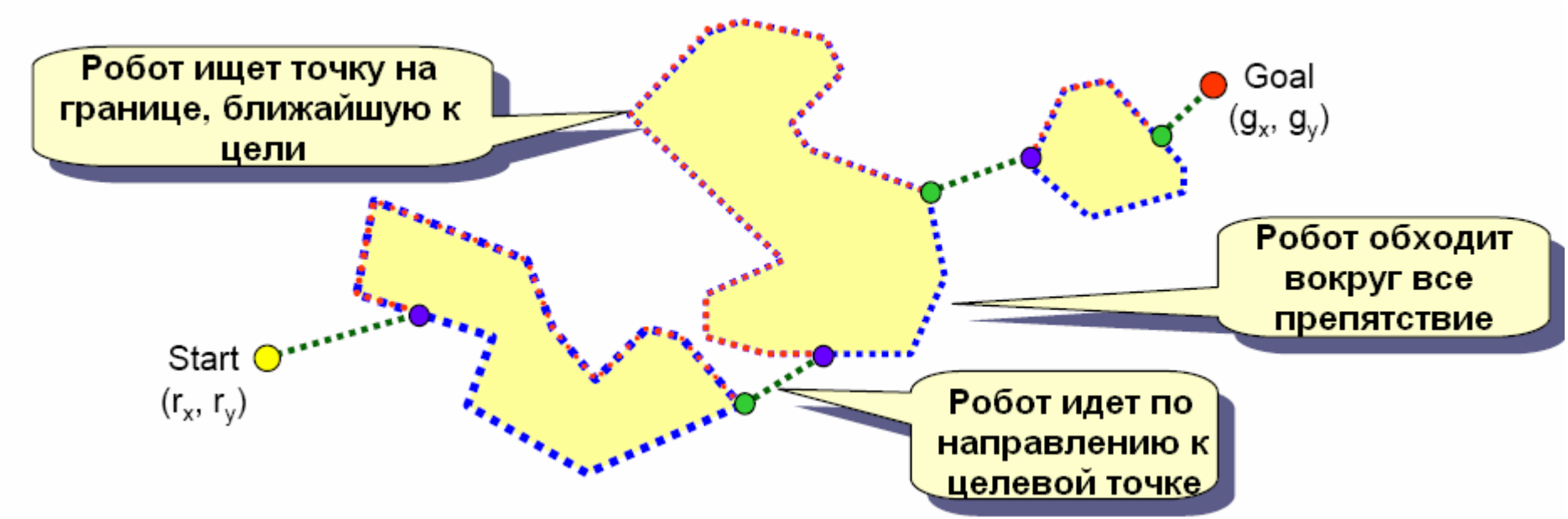
IV.3.1. Алгоритм Bug1

- Алгоритм Bug1 – первый и наиболее простой алгоритм из семейства Bug
- От робота требуется умение чувствовать «контакт» с препятствиями, т.е. расстояние для их распознавания может быть мало

IV.3.2. Алгоритм Bug1: стратегия

- Идем в сторону цели до тех пор, пока не встретим препятствие, запоминаем точку встречи
- Обходим препятствие вдоль его границы, пока не дойдем до точки встречи
- Если наиболее близкая точка не является точкой встречи препятствия, то идем к ней; иначе искомого пути к цели не существует

IV.3.3. Алгоритм Bug1: пример



IV.3.4. Алгоритм Bug1: анализ

- Алгоритм Bug1:
  - **Всегда** находит путь до целевой точки, если таковой существует
  - Проводит «всесторонний» поиск «наилучшей» точки для того, чтобы покинуть границу препятствия и направиться к цели
- Наибольшая дистанция, проходимая роботом равна:

$$d = r(S,G) + 1.5*[p(O_1) + \ldots + p(O_N)],$$

где:

$r(S,G)$ – расстояние от начала до цели

$p(O_K)$ – периметр K-ого препятствия

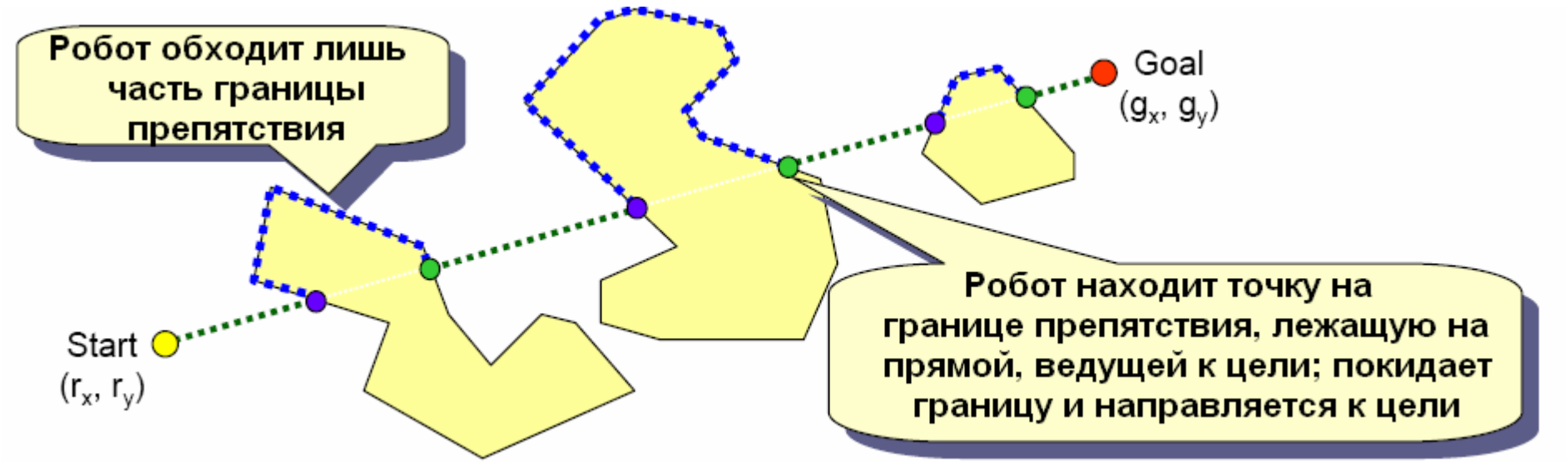
IV.4.1. Алгоритм Bug2

- Bug2 – модификация алгоритма Bug1, созданная с целью исключить необходимость обхода **всей** границы препятствия
- Как и в случае с Bug1, от робота требуется умение чувствовать «контакт» с препятствиями, т.е. расстояние для их распознавания может быть мало

IV.4.2. Алгоритм Bug2: стратегия

- Идем в сторону цели до тех пор, пока не встретим препятствие и запоминаем точку встречи и луч, ведущий к цели
- Обходим препятствие вдоль его границы до тех пор, пока не пересечем этот луч
- Если мы пришли не в точку встречи, продолжаем движение; иначе пути не существует

IV.4.3. Алгоритм Bug2: пример

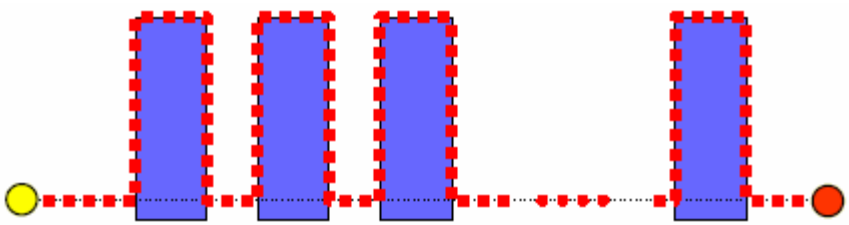


IV.4.4.1. Алгоритм Bug2: анализ

- Алгоритм Bug2:
  - **Всегда** находит путь до целевой точки, если таковой существует
  - Производит «жадный» поиск «наилучшей» точки для того, чтобы покинуть границу препятствия и направиться к цели

IV.4.4.2. Алгоритм Bug2: анализ

- Проходимая дистанция может существенно зависеть от направления, по которому робот обходит препятствие
- Худший случай (проходится практически весь периметр препятствий):



IV.4.4.3. Алгоритм Bug2: анализ

- Наибольшая дистанция, проходимая роботом равна:

  $d = d(S,G) + [(m_1/2 * p(O_1) + \ldots + + (m_N/2 * p(O_N)]$,

  где

  $d(S,G)$ – расстояние от начала до цели

  $p(O_K)$ – периметр K-ого препятствия

  m_i – число раз, которое луч от начала до цели пересекает i-ое препятствие

IV.5. Алгоритмы Bug1 и Bug2

- В общем случае Bug2 работает быстрее, чем Bug1, но существуют среды со сложными препятствиями, в которых Bug1 оказывается быстрее
- Плюсом обоих алгоритмов является требование умения робота чувствовать «контакт» лишь на малом расстоянии

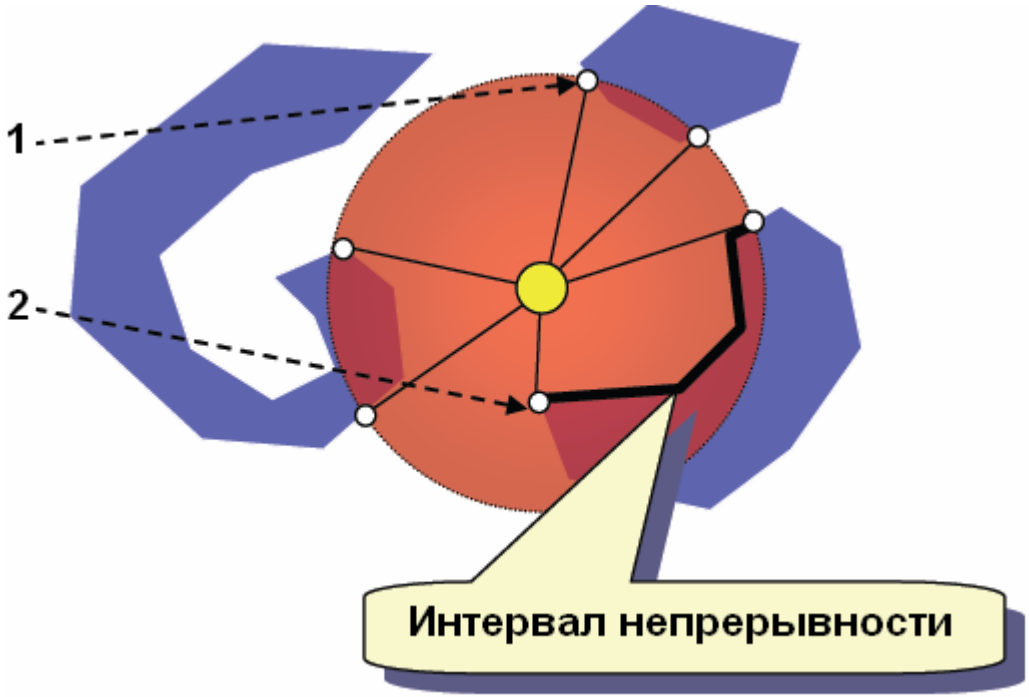
IV.6.1. Алгоритм Tangent Bug

- Алгоритм Tangent Bug был создан с целью улучшения алгоритмов Bug за счет оснащения робота дополнительными сенсорами, чтобы он мог ощущать препятствия вокруг себя на некотором расстоянии, которое сенсоры должны уметь определять
- На практике фиксируется угол между сенсорами (например, 5 градусов)

IV.6.2. Алгоритм Tangent Bug важные определения

- Введем два важных определения:
  - **Точка разрыва** – точка, в которой теряются показания сенсоров из-за:
    - Препятствия вне зоны сенсора (1)
    - Неопределенного препятствия (2)
  - **Интервал непрерывности** – интервал, определяемый двумя точками разрыва

IV.6.3. Алгоритм Tangent Bug пояснение к определениям



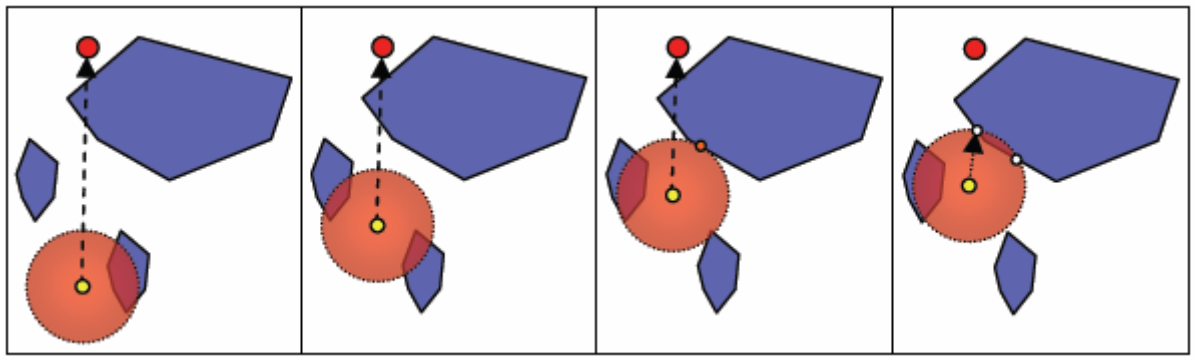
IV.6.4.1. Алгоритм Tangent Bug: стратегия

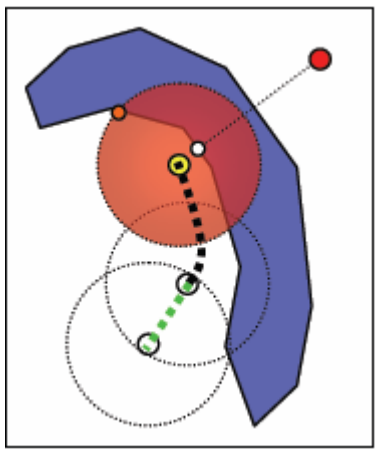
- Идем в сторону цели до тех пор, пока не почувствуем препятствие на прямой к цели; в случае нахождения препятствия прямая будет пересекать интервал непрерывности
- Движемся к одной из точек разрыва, для которой эвристическая оценка минимальна (например к той, сумма расстояний от которой до робота и до цели минимальна)

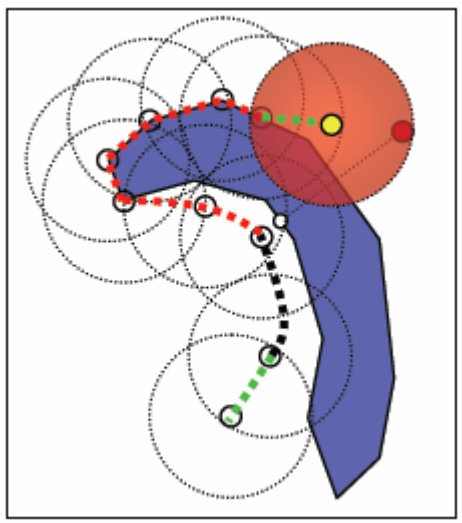
IV.6.4.2. Алгоритм Tangent Bug: стратегия

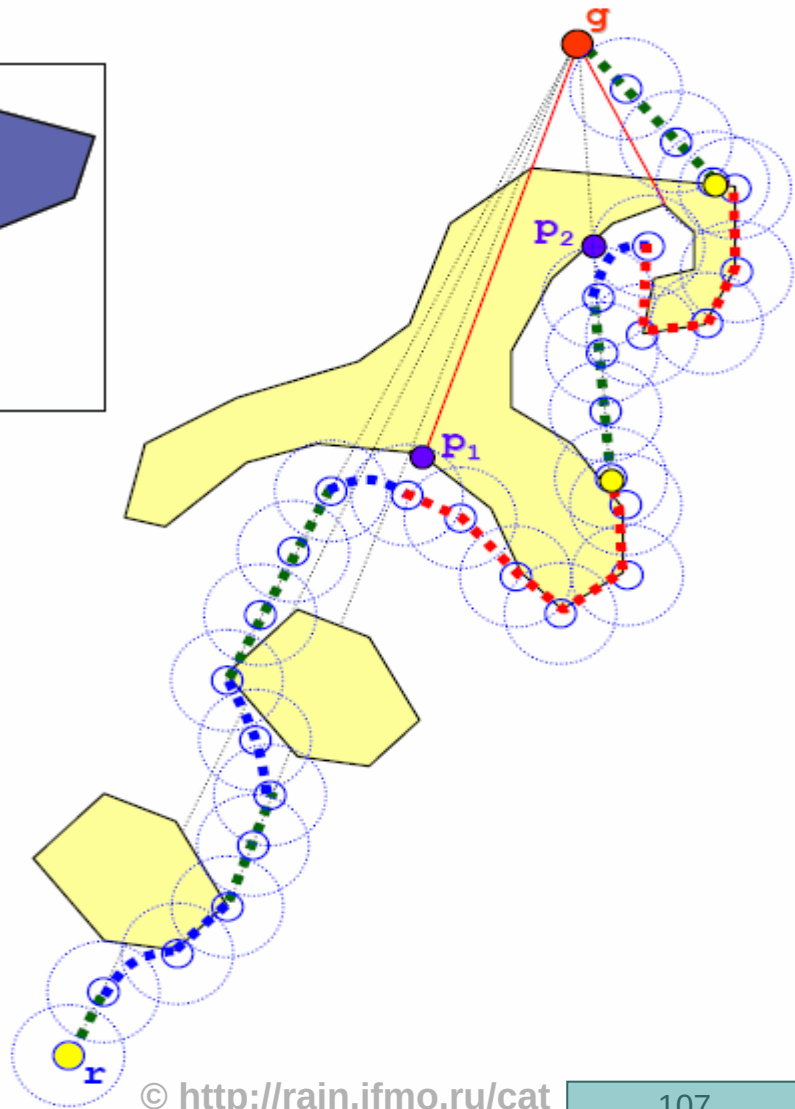
- В процессе движения получаем новые точки разрыва и движемся к ним до тех пор, пока эвристическая оценка не перестанет уменьшаться (т.е. мы достигаем локального минимума)
- Движемся вдоль границы препятствия, сохраняя направление
- Покидаем границу, когда препятствие больше не мешает проходу к целевой точке

IV.6.5. Алгоритм Tangent Bug: примеры









IV.6.6. Алгоритмы | заключение

- Преимущества алгоритмов Bug:
  - Простота и интуитивная понятность
  - Несложная реализация
  - Гарантированное (теоретически доказано) нахождение пути (если таковой существует)
- Недостатки на практике:
  - Невозможно идеальное позиционирование
  - Невозможно создать безошибочные сенсоры

V. Применение алгоритмов на практике

V.1.1. Применение в игровых проектах

- Применение в игровых проектах
  - Необходимы:
    - Быстрота работы
    - Универсальность работы (нахождение путей из любых точек)
  - Можно пренебречь:
    - Временем предрасчета
    - Точностью алгоритма

V.1.2. Стратегии в игровых проектах

- Стратегии
  - В основном используется метод потенциальных полей, с заранее рассчитанным полем всех препятствий в каждой точке карты; преимущества:
    - Быстрота нахождения пути из любых точек
    - Гладкость траектории

V.1.3. Применение в играх жанра Action

- Граф видимости
  - Удобство редактирования карты (быстрота)
  - Возможность вручную задавать более перспективные маршруты
  - Легкость исправления параметров карты для игры
- Комбинирование графа видимости и метода потенциальных полей

V.2.1. Применение алгоритмов в других областях

- Робототехника
  - Алгоритмы серии Bug с картами местности
    - Например, бытовые роботы-уборщики
  - Создание управляемых роботов, для навигации по картам со спутников
    - Используются алгоритмы для навигации по известной местности
    - При обнаружении неизвестных объектов используются алгоритмы семейства Bug

V.2.2. Применение алгоритмов в других областях

- Создание карт для морской и сухопутной навигации, автоматизация, создание систем автоматического управления транспортными средствами:
  - Метод потенциальных полей
  - Алгоритмы серии Bug

Литература

- Бондарев В.М., Рублинецкий В.И., Качко Е.Г. Основы программирования. – Харьков: Фолио; Ростов н/Д: Феникс, 1997
- B. Stout. Smart Moves: Intelligent Pathfinding. – (Перевод см. http://algolist.manual.ru/games/smartmove.php)
- M. Lanthier. Mobile Robot Positioning. – http://www.scs.carleton.ca/~lanthier/
- http://wikipedia.org
  - http://en.wikipedia.org/wiki/A*
  - http://en.wikipedia.org/wiki/Best-first_search
  - http://en.wikipedia.org/wiki/Taxicab_geometry