#### **Алгоритм Дейкстры**

Для понимания алгоритма Дейсктры познакомимся с основами графовой теории.

**Основные понятия теории графов**

**Граф** – математическая структура, используемая для представления множества объектов и отношений между ними, состоящая из двух компонентов: вершин и рёбер.

**Вершины** – это отдельные объекты в графах, в задачи поиска пути могут представлять физические места.

**Рёбра** – это связи между вершинами, они бывают взвешенными или невзвешенными, а также направленными или ненаправленными. В взвешенном графе каждому ребру присваивается значение, представляющее стоимость или расстояние между двумя вершинами. В направленном графе рёбра имеют направление, т.е. они ведут от одной вершины к другому.

**Путь в графе** – это последовательность вершин, в которой каждая следующая вершина, связана с предыдущей ребром. Длина пути – сумма весов всех рёбер, если граф взвешенный, или просто количество рёбер в ином случаи.

Алгоритмы поиска пути используют структуру графа и веса его рёбер, чтобы найти оптимальный путь между двумя вершинами. Оптимальным может быть самый дешёвый путь, самый короткий, и т.д.. То есть, оптимальность пути определяется исходя из задачи.

Алгоритм Дейкстры

Алгоритм Дейкстры - классический алгоритм нахождения кратчайшего пути в графах с неотрицательными весами. Используется как основа для многих других алгоритмов маршрутизации и планирования пути.

Алгоритм Дейкстры – это классический алгоритм поиска пути, который был разработан *Эдсгером Дейкстрой* в 1959 году. Он используется для поиска кратчайшего пути во взвешенном графе от одной вершины до всех остальных вершин.

Для начала устанавливается начальная вершина. Алгоритм Дейкстры работает по принципу “жадного ” алгоритма, т.е. на каждом шаге стримиться минимизировать текущую общую стоимость пути

Описание алгоритма:  
1. Инициализируются два множетсва:

A – множество обработанных вершин ( изначально пустое ) .

NA – множество всех остальных вершин ( изначально содержит все вершины графа ).

2. Каждой вершине присваивается вес – являющимся минимальной известной стоимости пути от начальной вершины до данной. Для начальной вершины присваиваемый вес равен 0, для остальных бесконечности.

3. Запускается процесс, где на каждом шаге из NA выбирается вершина с наименьшим весом, перемещается в А и обновляются веса всех соседей выбранной вершины. Вес соседа обновляется, если через выбранную вершину можно добраться до него с меньшей стоимостью. Процесс продолжается до того момента, как все вершины не будут посещены, либо не будет найден путь до конечной вершины, при условии, что она задана.

Алгоритм A\*

Алгоритм поиска пути с использованием эвристики.

Алгоритм А\* работает на основе оценки стоимости пути до цели, которая вычисляется как сумма двух компонент:

- известная стоимость пути от начальной вершины до текущей.  
 - эвристическая оценка стоимости пути от текущей вершины до цели.  
Эта сумма даёт оценку общей стоимости через данную вершину. Эвристика играет ключевую роль в алгоритме A\*. Она должна быть адекватной, чтобы алгоритм был эффективным.

Описание алгоритма:  
1. Алгоритм выбирает вершину с наименьшей оценкой из списка открытых вершин (вершин, которые уже были обнаружены, но еще не обработаны),

2. Смотрит на соседей этой вершины и обновляет их стоимости , если через текущую вершину можно добраться до них быстрее.

3. Если вершина еще не была открыта, она добавляется в список открытых вершин.

Процесс 1->2->3- продолжается, пока не будет найден путь до целевой вершины или пока не кончатся вершины в списке открытых вершин (что означает, что пути не существует).

Алгоритм D\* - это усовершенствованная версия алгоритма A\*, предназначенная для динамической перепланировки маршрута в меняющейся среде. Он эффективно находит оптимальный путь в реальном времени, даже если препятствия появляются или перемещаются во время движения.

Описание алгоритма:

В список открытых вершин добавляется конечная вершина.

Инициализируются значения и для каждого узла:

* + - известная стоимость кратчайшего пути от начальной точки до узла n.
  + - наименьшая стоимость пути до узла с учетом одного шага вперед.
  + – узлы, с которых можно перейти на – стоимость перехода с на .
  + - эвристическая функция, оценивающая расстояние от начальной точки до узла .

Для всех вершин значение устанавливается в бесконечность, а для конечной точки 0. Значение конечной точки устанавливается в 0.

**1. Основной цикл:**

Пока начальная точка не равна конечной точке или изменились условия

(например, появились препятствия):

* + - Из списка открытых вершин извлекается узел с наименьшим .
    - Если то:
      * Установите .
      * Для всех соседей узла n, кроме родителя:
        + Обновите значения .
        + Обновите ключи узлов и добавьте их в открытый список.
    - Если
      * Установите .
      * Обновите для узла n и его соседей.
      * Обновите ключи узлов и добавьте их в открытый список.

1. **Обновление графа**:
   * При обнаружении изменения в окружении (например, появление нового препятствия):
     + Обновите стоимость движения (weight) в измененных ячейках.
     + Пересчитайте для всех затронутых узлов.
     + Добавьте измененные узлы в открытый список с обновленными ключами.
2. **Планирование пути**:
   * Из текущей начальной точки следуйте по узлам, выбирая узел с наименьшим значением .
   * Если путь блокирован, переходите к шагу 2 для пересчета пути с учетом новых препятствий.

Картинки

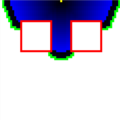
**Figure 1.** Расширение в процессе. Финишный узел (жёлтый) находится в середине верхнего ряда точек, начальный узел находится в середине нижнего ряда. Красный указывает на препятствие; чёрный/синий обозначает расширенные узлы (яркость обозначает стоимость). Зелёным обозначены узлы, которые расширяются.

Figure 1

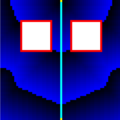
**Figure 2.** Расширение завершено. Путь указан голубым цветом**.**

Figure 2

RRT

Во многих случаях, когда мы планируем маршрут, нам нужно просто добраться из точки А в точку B, не зная ничего о предыдущих запросах планирования. Например, если окружающая среда меняется между запросами планирования, то часто более целесообразно будет перепланировать маршрут с самого начала, чем пытаться восстановить информацию из предыдущих запросов. Одним из методов, которые используются в таких ситуациях – это замена графа деревом.  
**Дерево** –этопросто особый тип графа, в котором узлы упорядочены в “родительские” и “дочерние”. Каждый узел имеет один родительский узел и ноль или более дочерних узлов.

Если мы будем хранить состояния, в которых был агент, в качестве родителей, а все состояния, в которые агент может перейти из них в качестве дочерних узлов, то мы сможем создать структуру в виде дерева, которое растёт из текущего состояния агента во все возможные места, в которых агент может находится. Рано или поздно дерево достигнет целевого состояния, которое нам нужно, и мы получим решение.

Задачу выращивания случайного дерева выполняют два класса алгоритмов:  
1. EST ( “Expansive Space Trees” или “расширяемые пространственные деревья” ), этот подход выбирает случайный узел в дереве и растёт в случайном направлении.   
2. RRT ( “Rapidly Exploring Randomized Trees”, “Быстрого исследования случайного дерева” )**,** этот подход начинается со случайной выборки узлов в пространстве, а затем выращивает ближайший узел по направлению к этой случайной выборке. Поговорим об RRT.

Описание алгоритма.

* Создаём пустое дерево T.
* Добавляем в Т начальное состояние агента.
* В течении N итераций, или пока не достигнута цель
  + Случайным образом делаем выборку для узла R.
  + Находим в Т ближайший к R узел. Называем его K.
  + Делаем шаги по лучу от К до R на малую величину эпсилон, пока не выполнится следующее условие:
    - При наличии коллизии возвращаемся к созданию случайной выборки.
    - В противном случае добавляем к Т новый узел в этой конфигурации.
    - Если мы достигли максимального расстояния δ от К, то возвращаемся к созданию случайной выборки.
* Если узел цели теперь находится в пределах расстояния от любого узла дерева, то у нас есть решение

Генетический алгоритм

GA – это эвристический метод решения оптимизационных задач, который был вдохновлён теорией Дарвина о естественном отборе. В нём решение задачи представляется в виде набора генов (свойств), которые вместе образуют хромосому (особь).

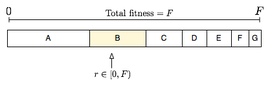
Описание алгоритма:

1. Создаётся начальное поколение с уникальными характеристиками. ( т.е. со случайно сгенерированными значениями исходного положения, скорости, ускорения и т.д.)
2. Отслеживает их достижение цели, выживание ( т.е. столкнулся ли робот с препятствием или нет ), итоговое удаление от цели, пройденное расстояние и вычисляет показатель приспособления ( функция, которая на основе некоторых данных определяет “хорошая” ли особь. Такими данными могут выступать: пройденный путь, удалённость от цели, столкновение с препятствием и т.д. )
3. На основе значения приспособления выбираются лучшие особи, которые будут использоваться для создания следующего поколения путем скрещивания и мутации. Этот процесс повторяется до достижения критерия остановки.
   * Скрещивание – процесс, при котором из хромосом родителей выбираются часть генов и создаётся новая особь, наследующая их.
   * Мутация – процесс, при котором “мутирует”какой-то ген хромосомы особи.

Критерий остановки может различаться в зависимости от задачи.

Алгоритм ACO разбит на 3 этапа. Необходимым условием для нахождения кратчайшего пути является инициализация и появление муравьев в графе в случайных узлах.

* Фаза 1 – Муравей “Прямого” поиска.
  + Перед всеми муравьями стоит задача достичь узла назначения за конечное число шагов. Если муравей достигает пункта назначения за ожидаемое количество шагов, он помечается как *пригодный*. На всех последующих этапах фокусируемся на пригодных муравьях.
  + На каждом шаге муравей определяет свой следующий ход, используя политику вероятности пограничного перехода (Transition Probability Policy  — вероятность перехода с текущего узла I на другой узел J)

Знаменатель уравнения означает все рёбра, по которым муравью разрешено передвигаться ( непосешённые рёбра ).  
Когда вероятности рёбер вычислены, муравей выбирает следующий узел на основе метода колеса рулетки. ( значения вероятностей перехода на узел нормализуются по формуле, , потом каждому значению присваивается “сектор” соответствующего размера и случайным образом выбирается число, по которому определяется узел).

* Фаза 2 – Муравей “Обратного” поиска.
  + Каждый пригодный муравей движется в обратном направлении от места назначения к месту появления, выделяя феромоны на этом пути.
  + Количество феромонов, выделяемого каждым муравьем на ребро рассчитывается по политике депозита феромонов (Pheromone Deposit Policy)

* Фаза 3 – Муравей “решения”.
  + На данном этапе были размещены несколько волн(эпох) “прямых“ и “обратных” муравьёв. Предполагается, что уровни феромонов по рёбрам *достаточно* велики, чтобы точно найти кратчайший маршрут от любого узла к пункту назначения.
  + Чтобы *сгенерировать* решение, т.е. кратчайший путь от начального узла до целевого, мы развертываем муравья “Решения” на начальном узле.
  + Уникальная характеристика муравья “Решения*”*, это его жадность к феромонам (), т.е. он будет выбирать следующий шаг, основываясь на ребро с наибольшим значением феромона, игнорируя эвристическую оценку.
  + Как только муравей “Решения” достигнет конечного узла, мы можем взять путь, пройденный им, в качестве решения ( т.е. кратчайшего пути ).
* **Полная информация о препятствиях:** Алгоритм А\* является информированным алгоритмом поиска, который использует знание о структуре пространства для эффективного поиска пути. В случае с задачей данной работы, изначально доступна вся информация о препятствиях, что позволяет алгоритму А\* эффективно искать путь, минимизируя стоимость пути и избегая препятствий.
* **Известны начальная и конечная точки:** Алгоритм А\* использует эвристику для оценки расстояния от текущей точки до конечной точки, что позволяет ему находить оптимальный путь. Так как начальная и конечная точки известны заранее, алгоритм А\* может эффективно использовать эту информация для поиска пути.
* **Статичная среда:** Алгоритм А\* хорошо работает в статических средах, где препятствия и цель не меняются во время поиска. В нашей задаче, среда является статичной, что делает алгоритм А\* хорошим выбором.
* **Оптимальность:** Алгоритм А\* гарантирует нахождения оптимального, если эвристика не переоценивает расстояние до цели. Формально говоря, это означает, что эвристика удовлетворяет условию , где – оценка стоимости до цели из узла , а – фактическая стоимость до цели.
* **Эффективность:** Алгоритм А\* довольно эффективен в терминах использовании памяти и времени.
* **Простота реализации и настройки:** Алгоритм A\* относительно прост в реализации и не требует сложных структур данных, что позволяет легче адаптировать алгоритм для решения нужной задачи

В данной главе будут описаны кодовая реализация алгоритмов, среда для тестирования и симулирования, а также будут показаны полученные результаты симулирования.

#### **2.1.1. Описание среды Webots**

Webots – это среда моделирования робототехники. Она предлагает обширный набор инструментов для проектирования робототехнических систем и проведения экспериментов в виртуальных средах.

**Основные возможности**

* **Моделирование роботов:** Webots позволяет пользователем создавать детализированные трёхмерные модели роботов с использованием языка описания роботов RDL ( Robot Description Language). Модели могут включать разные типы датчиков и актуаторов, а также сложные механизмы, такие как манипуляторы и колёса. Также можно использовать уже готовые модели, предоставляемые после установки среды.
* **Физическое моделирование:** Webots использует физический движок для точного моделирования динамики роботов и их взаимодействия с окружающей средой, что позволяет проводить эксперименты в условиях, близких к реальным.
* **Программирование роботов:** Webots поддерживает множество языков программирования, включая С, С++, Java, Python, что позволяет пользователям разрабатывать программы управления роботами и использовать существующие библиотеки и инструменты.
* **Визуализация: Webots** предлагает инструменты визуализации, которые позволяют отслеживать поведение роботов во время экспериментов. Кроме того, есть возможность записи хода эксперимента для последующего анализа и воспроизведения.

**Преимущества Webots**

Webots обладает рядом преимуществ:

1. Он предлагает гибкую и мощную среду для моделирования роботов и проведения экспериментов.
2. Он поддерживает широкий спектр робототехнических платформ и датчиков, что позволяет пользователям моделировать и тестировать реальные робототехнические системы.
3. Он является открытым проектом и активно поддерживается сообществом, что обеспечивает его постоянное развитие и улучшение

В целом, Webots – мощный инструмент, который может значительно упростить и ускорить процесс разработки и тестирования робототехнических систем. Более подробная информация про данную среду доступна на их официальном веб-сайте, документации и сообществе GitHub ([[[1]](#footnote-1)][[[2]](#footnote-2)][[[3]](#footnote-3)]).

#### **2.1.2. Реализация**

В качестве среды для моделирования была выбрана прямоугольная область RectangleArena, доступная в Webots (Figure 11). В качестве препятствий будут выступать ящики разных размеров расположенные внутри области в произвольном порядке (Figure 12). В качестве робота была выбрана модель Epuck ( Figure 13), а в качестве указателя целевой точки был выбран объект Ball ( Figure 14 ).

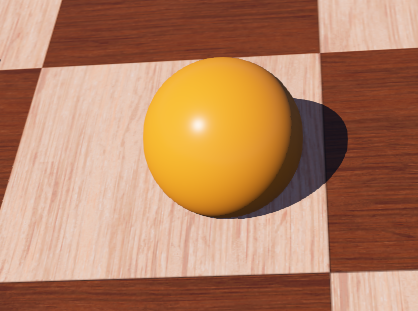
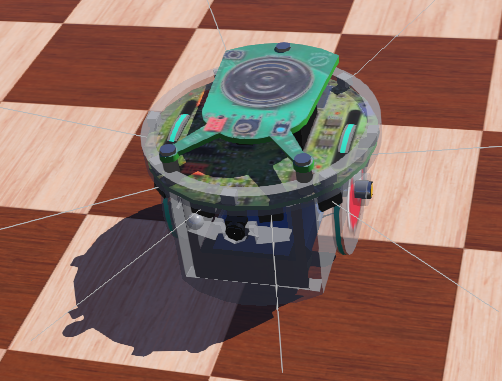
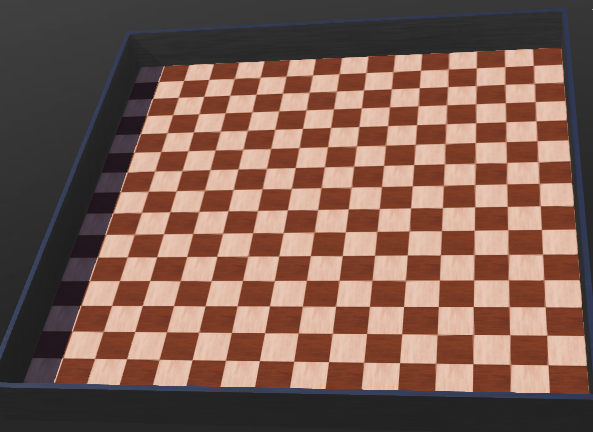


Figure 11 Figure 12 Figure 13 Figure 14

Сначала определим какие данные доступны изначально:

1. Координаты начального положения робота, его угол поворота.
2. Координаты препятствий, их углы поворотов.
3. Координаты целевой точки.
4. Размеры среды.

**Анализ программного кода для управления роботов в симуляционной среде**

Программная реализация кода разделена на несколько частей:

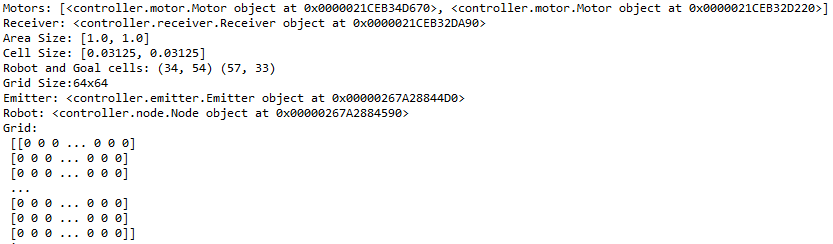
1. Инициализация объектов.
2. Дискретизация (Создание сетки).
3. Использование алгоритма А\* ( RRT ), для нахождения пути.
4. На основе полученного пути отправление команд на приёмник (receiver) робота для достижения цели (команды отправляются передатчиком ( emitter ) ).

1-3 части выполняются на устройстве передатчике, а само управление осуществляет приёмник, на основе команд, получаемых от передатчика в режиме реального времени.

##### Инициализация объектов.

На данном этапе инициализируются все объекты присутствующие в среде симулирования, т.е.:

* Робот ( Epuck ), его начальное положение, моторы.
* Цель ( Ball ), его координаты.
* Арена ( RectangleArena ), его размеры (ArenaSize), а также на основе размеров арены выбирается размер ячейки ( CellSize ) для дальнейшей дискретизации и представления в виде сетки (см. [Дискретизация](#_Дискретизация) ).
* Препятствия ( BoxN, где N – номер препятствия ), координаты их угловых сторон, не учитывая поворот ( поворот учитывается на этапе создания сетки: см. [Дискретизация](#_Дискретизация) ), а также угол поворота препятствия.
* Передатчик ( emitter ) и приёмник ( receiver ).
* Инициализируется пустая двумерная матрица ( далее сетка ), которая будет использоваться при дискретизации. Размер матрицы – , где
* Т.к. все координаты получаются в виде числа с плавающей запятой, то находится их положения в сетке с использованием формулы (т.к. точка – центр арены):

Figure 15: Процесс инициализации объектов и переменных.

##### *Дискретизация*

После инициализации всех значений, начинается процесс дискретизации, т.е. представления всей среды в виде двумерной матрицы ( Grid ).

На шаге инициализации уже были найдены и – дискретные индексы робота и целевого объекта в сетке, которые будут в дальнейшем использоваться при этапе вызове алгоритма. На этапе инициализации значений были найдены координаты углов препятствий, не учитывая поворот. На данном этапе вычисляются координаты после поворота с использованием формулы:

, где   
 – координаты после поворота.  
и приводятся к дискретному виду по указанной в прошлом этапе формуле. После нахождения индексов углов, они соединяются с помощью алгоритма Брезенхэма [[[4]](#footnote-4)], а элементам соответствующим индексам в сетке присваивается значение -1. ( Пример дискретизации области среды: см. Figure 15-16).

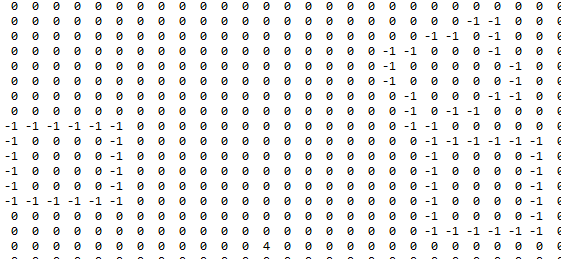


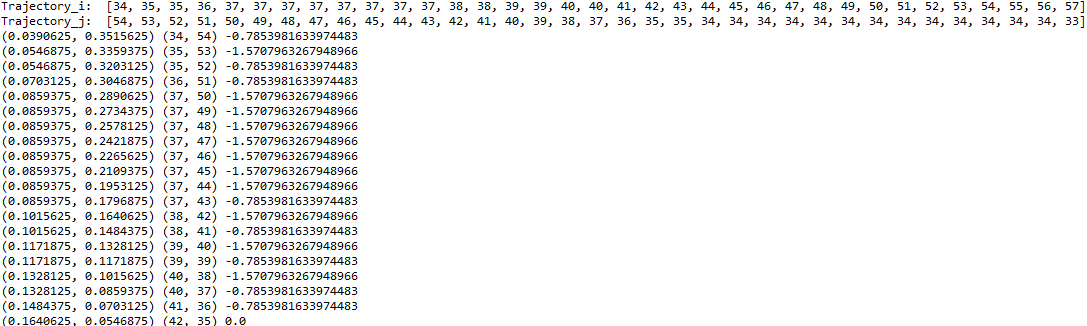
Figure 15 Figure 16

##### *Вызов алгоритма*

На основе данных, полученных на прошлых этапах, вызывается алгоритм А\* ( RRT ), который возвращает траекторию пути в виде списков индексов сетки.

Полученные индексы обратно преобразовываются в представление чисел с плавающей запятой с помощью формул представленных ниже:

Также находятся углы движения, от , которые будут использоваться при управлении роботом, на следующем этапе.

Figure 17: Вычисление

##### *Управление роботом*

В итоге предыдущих этапов были получены координаты траектории и повороты, которые необходимо совершать на них, теперь необходимо управлять роботом, используя эти данные. За обработку этих данных и отправление команд роботу отвечает передатчик. Команды делятся на 4 типа: “остановка”, “поворот по часовой стрелке”, “поворот против часовой стрелки”, “движение вперёд”. Чтобы понять, какую команду надо отправить, передатчик берёт текущие координаты робота, его угол поворота и сравнивает с ожидаемыми координатами и углом, которые были получены на прошлом этапе. Для сравнения используется Евклидово расстояние. Робот полностью останавливается, когда расстояние от него до цели становится меньше какого-то малого числа .

Робот управляется следующим образом.

1. На первом шаге, робот начинает двигаться по прямо от своей стартовой позиции до первой найденной точки траектории.
2. Когда расстояние между текущим положением робота и точкой траектории становится меньше малого числа, то последовательно отправляются команда остановки и команда поворота к следующей точке.
3. Когда угол поворота робота становится достаточно близкой к требуемом, то отправляется команда остановки, потом команда движения вперёд.
4. Пункты 2-3 повторяются до тех пор, пока расстояние между последней точкой траектории ( т.е. целью ) и текущим положением робота не будут меньше какого-то малого числа .

В пункте 2 также вычисляется какая команда поворота будет отправлена – “поворот против часовой стрелки” или “поворот по часовой стрелке”.  
Вычисление происходит следующим образом:

На последнем этапе передатчик на основе полученной пути отправляет команды приёмнику робота, до достижения конечной точки.

Робот управляется следующим образом.

1. На первом шаге, робот начинает двигаться по прямо от своей стартовой позиции до первой найденной точки траектории.
2. Когда расстояние между текущим положением робота и точкой траектории становится меньше малого числа, то последовательно отправляются команда остановки и команда поворота к следующей точке.
3. Когда угол поворота робота становится достаточно близкой к требуемому, то отправляется команда остановки, потом команда движения вперёд.
4. Пункты 2-3 повторяются до тех пор, пока расстояние между последней точкой траектории ( т.е. целью ) и текущим положением робота не будут меньше какого-то малого числа .

1. **. Официальный сайт Webots:** <https://cyberbotics.com/> [↑](#footnote-ref-1)
2. **. Документация Webots:** <https://cyberbotics.com/doc/guide/index> [↑](#footnote-ref-2)
3. **. Сообщество Webots:** <https://github.com/cyberbotics/webots> [↑](#footnote-ref-3)
4. . **Роджерс Д. Алгоритмические основы машинной графики. — М.: Мир, 1989. — С. 54-63. — ISBN 5-03-000476-9.** [↑](#footnote-ref-4)