



Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное  
образовательное учреждение высшего образования  
«Московский государственный технический университет  
имени Н.Э. Баумана  
(национальный исследовательский университет)»  
(МГТУ им. Н.Э. Баумана)

---

ФАКУЛЬТЕТ \_\_\_\_\_ «Информатика и системы управления»  
КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

## Отчет по лабораторной работе №6 по курсу "Анализ алгоритмов"

Тема Муравьиный алгоритм

Студент Якуба Д. В.

Группа ИУ7-53Б

Оценка (баллы) \_\_\_\_\_

Преподаватели Волкова Л.Л., Строганов Ю.В.

Москва — 2020 г.

# Оглавление

<b>Введение</b>	<b>3</b>
<b>1 Аналитическая часть</b>	<b>4</b>
1.1 Задача коммивояжера . . . . .	4
1.2 Алгоритм полного перебора для решения задачи коммивояжера . . . . .	4
1.3 Муравьиный алгоритм для решения задачи коммивояжера	5
<b>2 Конструкторская часть</b>	<b>7</b>
2.1 Схема алгоритмы полного перебора . . . . .	7
2.2 Схема муравьиного алгоритма . . . . .	9
<b>3 Технологическая часть</b>	<b>12</b>
3.1 Требования к программному обеспечению . . . . .	12
3.2 Средства реализации программного обеспечения . . . . .	12
3.3 Листинг кода . . . . .	13
3.4 Тестирование программного продукта . . . . .	17
<b>4 Исследовательская часть</b>	<b>19</b>
4.1 Технические характеристики . . . . .	19
4.2 Пример работы программного обеспечения . . . . .	19
4.3 Время выполнения алгоритмов . . . . .	21
<b>Заключение</b>	<b>23</b>
<b>Литература</b>	<b>23</b>

# Введение

## Цель лабораторной работы

Реализация муравьиного алгоритма и приобретение навыков параметризации методов на примере реализованного алгоритма, примененного к задаче коммивояжера.

## Задачи лабораторной работы

- 1) изучить алгоритм полного перебора для решения задачи коммивояжера;
- 2) реализовать алгоритм полного перебора для решения задачи коммивояжера;
- 3) изучить муравьиный алгоритм для решения задачи коммивояжера;
- 4) реализовать муравьиный алгоритм для решения задачи коммивояжера;
- 5) провести сравнительный анализ скорости работы реализованных алгоритмов;
- 5) подготовить отчёт по проведенной работе.

# 1 | Аналитическая часть

В данном разделе описаны задача коммивояжёра, идея муравьиного алгоритма и алгоритма полного перебора для решения этой задачи.

## 1.1 Задача коммивояжера

Коммивояжёр (фр. *commis voyageur*) — бродячий торговец. Задача коммивояжёра — важная задача транспортной логистики, отрасли, занимающейся планированием транспортных перевозок [1]. В описываемой задаче рассматривается несколько городов и матрица попарных расстояний между ними. Требуется найти такой порядок посещения городов, чтобы суммарное пройденное расстояние было минимальным, каждый город посещался ровно один раз и коммивояжер вернулся в тот город, с которого начал свой маршрут. Другими словами, во взвешенном полном графе требуется найти гамильтонов цикл минимального веса.

## 1.2 Алгоритм полного перебора для решения задачи коммивояжера

Алгоритм полного перебора для решения задачи коммивояжера предполагает рассмотрение всех возможных путей в графе и выбор наименьшего из них.

Такой подход гарантирует точное решение задачи, однако, так как задача относится к числу трансвычислительных [2], то уже при небольшом числе городов решение за приемлемое время невозможно.

### 1.3 Муравьиный алгоритм для решения задачи коммивояжера

Муравьиные алгоритмы представляют собой новый перспективный метод решения задач оптимизации, в основе которого лежит моделирование поведения колонии муравьев [3]. Колония представляет собой систему с очень простыми правилами автономного поведения особей.

Каждый муравей определяет для себя маршрут, который необходимо пройти на основе феромона, который он ощущает, во время прохождения, каждый муравей оставляет феромон на своем пути, чтобы остальные муравьи могли по нему ориентироваться. В результате при прохождении каждым муравьем различного маршрута наибольшее число феромона остается на оптимальном пути.

Самоорганизация колонии является результатом взаимодействия следующих компонентов:

- случайность — муравьи имеют случайную природу движения;
- многократность — колония допускает число муравьев, достигающее от нескольких десятков до миллионов особей;
- положительная обратная связь — во время движения муравей откладывает феромон, позволяющий другим особям определить для себя оптимальный маршрут;
- отрицательная обратная связь — по истечении определенного времени феромон испаряется;
- целевая функция.

Пусть муравей обладает следующими характеристиками:

- зрение — определяет длину ребра;
- обоняние — чувствует феромон;
- память — запоминает маршрут, который прошел.

Введем целевую функцию  $\eta_{ij} = 1/D_{ij}$ , где  $D_{ij}$  — расстояние из текущего пункта  $i$  до заданного пункта  $j$ .

Посчитаем вероятности перехода в заданную точку по формуле (1.1):

$$P_{kij} = \begin{cases} \frac{t_{ij}^a \eta_{ij}^b}{\sum_{q=1}^m t_{iq}^a \eta_{iq}^b}, & \text{вершина не была посещена ранее муравьем } k, \\ 0, & \text{иначе} \end{cases} \quad (1.1)$$

где  $a, b$  – настраиваемые параметры,  $t$  – концентрация феромона, причем  $a + b = \text{const}$ , а при  $a = 0$  алгоритм вырождается в жадный [?].

Когда все муравьи завершили движение происходит обновление феромона по формуле (1.2):

$$t_{ij}(t+1) = (1-p)t_{ij}(t) + \Delta t_{ij}, \Delta t_{ij} = \sum_{k=1}^N t_{ij}^k \quad (1.2)$$

где

$$\Delta t_{ij}^k = \begin{cases} Q/L_k, & \text{ребро посещено } k\text{-ым муравьем,} \\ 0, & \text{иначе} \end{cases} \quad (1.3)$$

$L_k$  — длина пути  $k$ -ого муравья,  $Q$  — настраивает концентрацию нанесения/испарения феромона,  $N$  — количество муравьев.

## Вывод

Были рассмотрены задача коммивояжера, муравьиный алгоритм и алгоритм полного перебора для решения поставленной задачи.

В данной работе стоит задача реализации двух рассмотренных алгоритмов.

## 2 | Конструкторская часть

В данном разделе представлены схемы муравьиного алгоритма и алгоритма полного перебора для решения задачи коммивояжера.

### 2.1 Схема алгоритмы полного перебора

Схема алгоритма Брезенхема предоставлена на рисунках 2.1, 2.2.

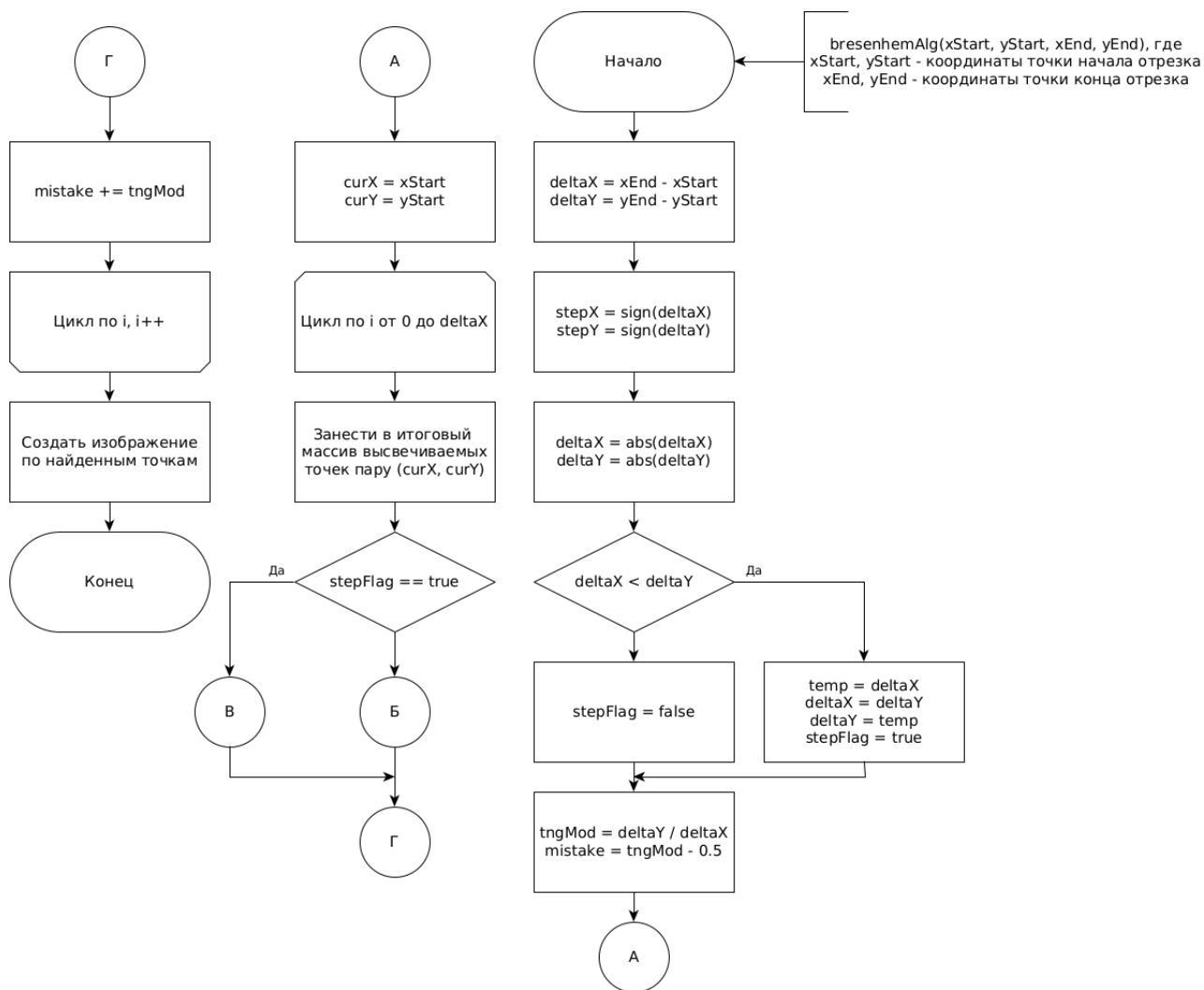


Рис. 2.1: Схема алгоритма Брезенхема.



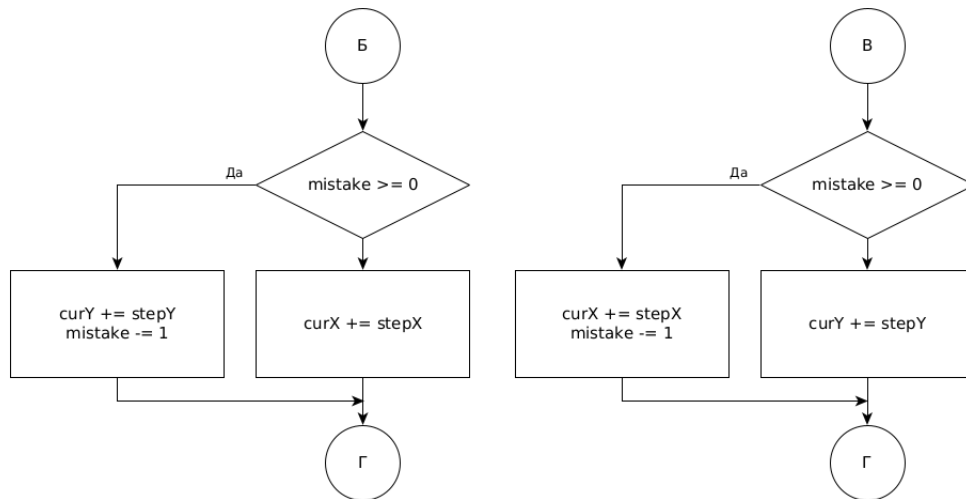


Рис. 2.2: Схема алгоритма Брезенхема.

## 2.2 Схема муравьиного алгоритма

На рисунках 2.3, 2.4 предоставлена IDEF0 схема реализации конвейерной обработки данных для алгоритма Брезенхема.

Как видно из схемы уровня декомпозиции 1 (2.4) задействуется 3 потока. Поток №1 решает задачу подготовки данных для растрирования отрезка. Поток №2, задействуя входные данные и данные, полученные от потока №1, заполняет массив точек, наилучшим образом аппроксимирующих отрезок целочисленными координатами экрана пользователя. Поток №3, опираясь на данные, полученные от потока №2, непосредственно создает объект, который и будет в дальнейшем визуализирован.

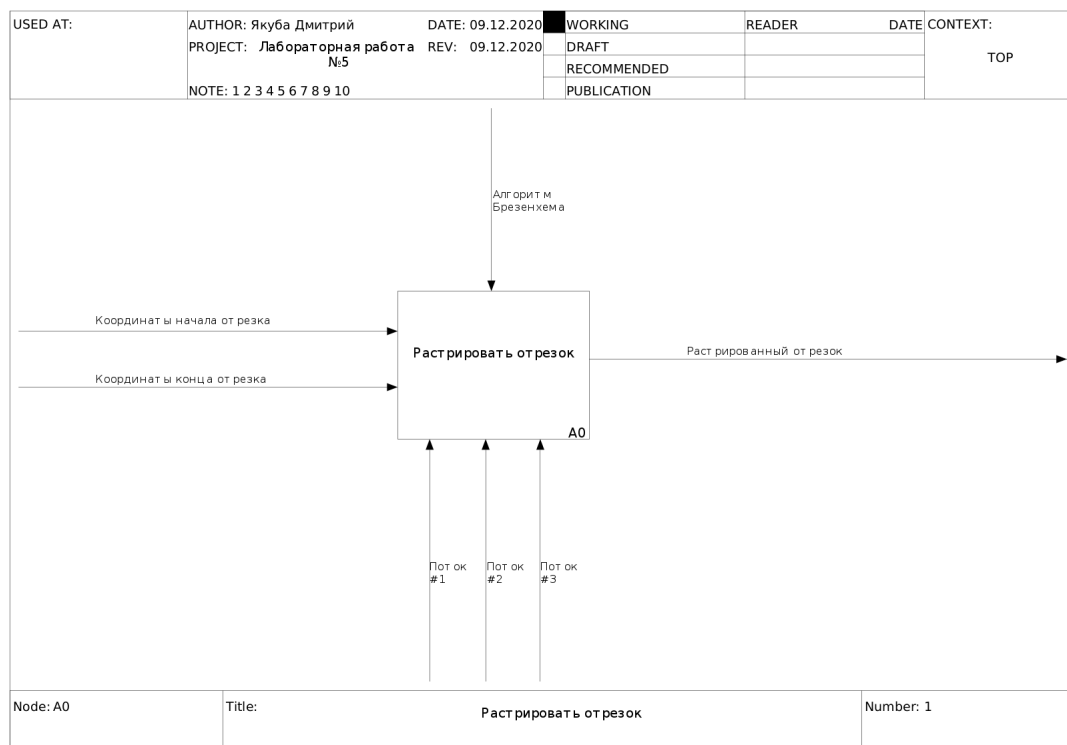


Рис. 2.3: Схема реализации конвейерной обработки данных для алгоритма Брезенхема.

## Вывод

Были представлены схемы алгоритма Брезенхема, а также реализации конвейерной обработки данных для данного алгоритма.

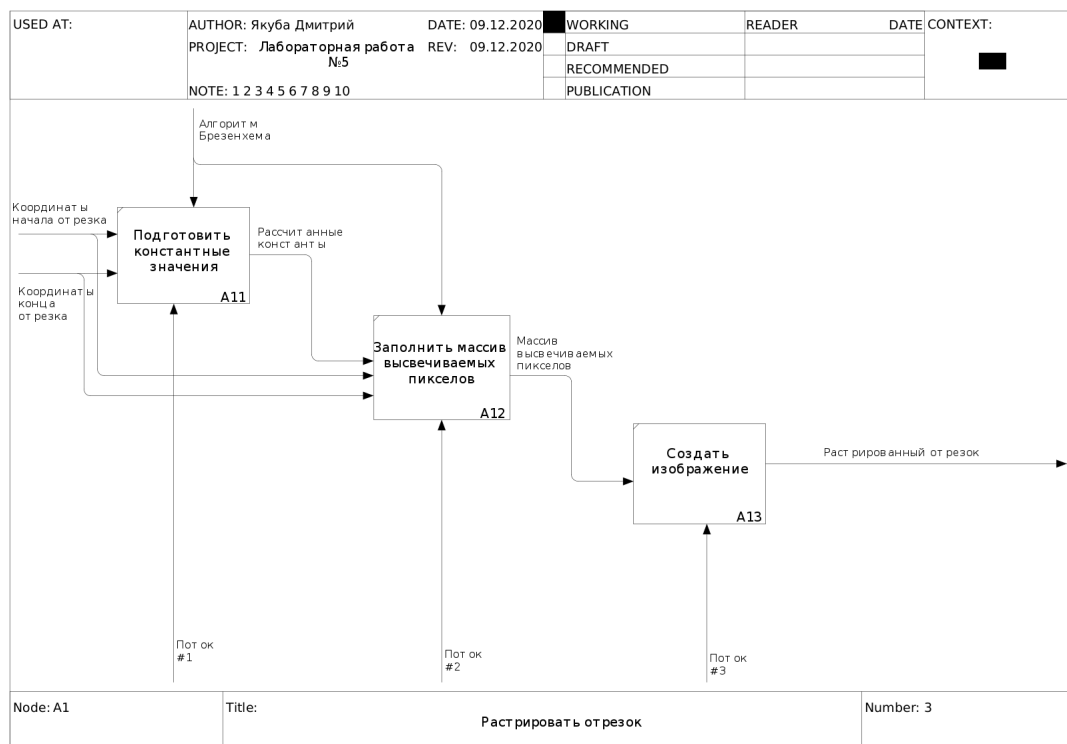


Рис. 2.4: Схема реализации конвейерной обработки данных для алгоритма Брезенхема.

## 3 | Технологическая часть

В данном разделе приведены требования к программному обеспечению, средства реализации программного обеспечения, а также листинг кода.

### 3.1 Требования к программному обеспечению

- входные данные - количество выполняемых задач (количество rasterизуемых отрезков);
- выходные данные - записи времени прихода и ухода обрабатываемых заявок для каждого реализованного конвейера.

### 3.2 Средства реализации программного обеспечения

При написании программного продукта был использован язык программирования C++ [4].

Данный выбор обусловлен следующими факторами:

- данный язык программирования преподавался в рамках курса объектно-ориентированного программирования;
- высокая вычислительная производительность;
- большое количество справочной и учебной литературы в сети Интернет;
- наличие реализации нативных потоков.

При написании программного продукта использовалась среда разработки QT Creator [5].

Данный выбор обусловлен следующими факторами:

- основы работы с данной средой разработки преподавался в рамках курса программирования на Си;
- QT Creator позволяет работать с расширением QtDesign, позволяющим создавать визуализируемый объект.

Для проведения замеров времени использовалась сторонняя библиотека Boost [6]. Данная библиотека позволила фиксировать время прихода и ухода каждой заявки с точностью до наносекунд.

### 3.3 Листинг кода

В листингах 3.1 и 3.2 предоставлены реализации рассматриваемых алгоритмов.

Листинг 3.1: Разбиение алгоритма Брезенхема

```
1 std::string now_str()
2 {
3     const boost::posix_time::ptime now = boost::posix_time::
        microsec_clock::local_time();
4
5     const boost::posix_time::time_duration td = now.time_of_day()
        ;
6
7     const long hours = td.hours();
8     const long minutes = td.minutes();
9     const long seconds = td.seconds();
10    const long nanoseconds =
11        td.total_nanoseconds() - ((hours * 3600 + minutes * 60 +
            seconds) * 1000000000);
12
13    char buf[40];
14    sprintf(buf, "%02ld:%02ld:%02ld.%09ld", hours, minutes,
        seconds, nanoseconds);
15
16    return buf;
17 }
18
```

```

19 SegmentRasterizator::SegmentRasterizator(int xStart_, int yStart_
    , int xEnd_, int yEnd_)
20 {
21     xStart = xStart_;
22     yStart = yStart_;
23     xEnd = xEnd_;
24     yEnd = yEnd_;
25     if (xStart == xEnd)
26         xEnd += 1;
27     else if (yStart == yEnd)
28         yEnd += 1;
29
30     image = new QImage(WIDTH, HEIGHT, QImage::Format_RGB32);
31     image->fill(Qt::white);
32 }
33
34 int sign(float num)
35 {
36     return (num < __FLT_EPSILON__) ? -1 : ((num >
        __FLT_EPSILON__) ? 1 : 0);
37 }
38
39 void SegmentRasterizator::prepareConstantsForRB(int index)
40 {
41     std::printf(ANSI_BLUE_BRIGHT "From START worker: task %d
        BEGIN %s" ANSI_RESET "\n", index, now_str().c_str());
42
43     deltaX = xEnd - xStart;
44     deltaY = yEnd - yStart;
45
46     stepX = sign(deltaX);
47     stepY = sign(deltaY);
48
49     deltaX = std::abs(deltaX);
50     deltaY = std::abs(deltaY);
51
52     if (deltaX < deltaY)
53     {
54         std::swap(deltaX, deltaY);
55         stepFlag = true;
56     }
57     else
58         stepFlag = false;
59
60     tngModule = deltaY / deltaX;

```

```

61     mistake = tngModule - 0.5;
62
63     std::printf(ANSI_BLUE_BRIGHT "From START worker: task %d
        ENDED %s" ANSI_RESET "\n", index, now_str().c_str());
64 }
65
66 void SegmentRasterizator::rastSegment(int index)
67 {
68     std::printf(ANSI_MAGENTA_BRIGHT "From MIDDLE worker: task %d
        BEGIN %s" ANSI_RESET "\n", index, now_str().c_str());
69
70     float curX = xStart, curY = yStart;
71     for (int i = 0; i <= deltaX; i++)
72     {
73         dotsOfSegment.push_back(std::pair<int, int>(curX, curY));
74         if (stepFlag)
75         {
76             if (mistake >= 0)
77                 (curX += stepX, mistake--);
78             curY += stepY;
79         }
80         else
81         {
82             if (mistake >= 0)
83                 (curY += stepY, mistake--);
84             curX += stepX;
85         }
86         mistake += tngModule;
87     }
88
89     std::printf(ANSI_MAGENTA_BRIGHT "From MIDDLE worker: task %d
        ENDED %s" ANSI_RESET "\n", index, now_str().c_str());
90 }
91
92 void SegmentRasterizator::createImg(int index)
93 {
94     std::printf(ANSI_CYAN_BRIGHT "From END worker: task %d BEGIN %
        s" ANSI_RESET "\n", index, now_str().c_str());
95
96     for (auto iter = dotsOfSegment.begin(); iter < dotsOfSegment.
        end(); iter++)
97         image->setPixel(iter->first, iter->second, Qt::black);
98
99     std::printf(ANSI_CYAN_BRIGHT "From END worker: task %d ENDED
        %s" ANSI_RESET "\n", index, now_str().c_str());

```

```

100 }
101
102 std::vector<std::pair<int, int>> SegmentRasterizator::
    getDotsOfSegment()
103 {
104     return dotsOfSegment;
105 }

```

Листинг 3.2: Менеджер потоков

```

1 Director::Director(std::queue<SegmentRasterizator> &startQueue_)
2 {
3     startQueue = startQueue_;
4 }
5
6 void Director::processPrepare()
7 {
8     int i = 0;
9     for (SegmentRasterizator curSeg(startQueue.front());
10         startQueue.size();
11         startQueue.pop(), curSeg = startQueue.front())
12     {
13         curSeg.prepareConstantsForRB(i++);
14         middleQueue.push(curSeg);
15     }
16 }
17
18 void Director::processRast()
19 {
20     int i = 0;
21     while (startQueue.size() || middleQueue.size())
22     {
23         if (middleQueue.empty())
24             continue;
25         SegmentRasterizator curSeg(middleQueue.front());
26
27         curSeg.rastSegment(i++);
28
29         endQueue.push(curSeg);
30         middleQueue.pop();
31     }
32 }
33
34 void Director::processCreate()
35 {

```



```

36     int i = 0;
37     while (startQueue.size() || middleQueue.size() || endQueue.
        size())
38     {
39         if (endQueue.empty())
40             continue;
41         SegmentRasterizator curSeg(endQueue.front());
42
43         curSeg.createImg(i++);
44         endQueue.pop();
45         final.push_back(curSeg);
46     }
47 }
48
49 void Director::initWork()
50 {
51     workers[0] = std::thread(&Director::processPrepare, this);
52     workers[1] = std::thread(&Director::processRast, this);
53     workers[2] = std::thread(&Director::processCreate, this);
54
55     workers[0].join();
56     workers[1].join();
57     workers[2].join();
58 }
59
60 std::vector<SegmentRasterizator> Director::getFinal() { return
    final; }

```

### 3.4 Тестирование программного продукта

В таблице 3.1 приведены тесты для функций, реализующих алгоритм Брезенхема. Тесты пройдены успешно.

Таблица 3.1: Тестирование функций

Точка начала отрезка (x, y)	Точка конца отрезка (x, y)	Ожидаемый результат
(1, 1)	(3, 3)	(1, 1), (2, 2), (3, 3)
(1, 1)	(1, 3)	(1, 1), (1, 2), (1, 3)
(1, 1)	(2, 1)	(1, 1), (2, 1)
(3, 3)	(1, 1)	(1, 1), (2, 2), (3, 3)

## Вывод

Спроектированные алгоритмы были реализованы и протестированы.

## 4 | Исследовательская часть

### 4.1 Технические характеристики

Технические характеристики ЭВМ, на котором выполнялись исследования:

- ОС: Manjaro Linux 20.1.1 Mikah;
- Оперативная память: 16 Гб;
- Процессор: Intel Core i7-10510U.

При проведении замеров времени ноутбук был подключен к сети электропитания.

### 4.2 Пример работы программного обеспечения

На рисунке 4.1 приведен пример работы программы для 7 визуализируемых отрезков.

```
From START worker: task 0 BEGIN 16:11:30.449847000
From START worker: task 0 ENDED 16:11:30.449944000
From START worker: task 1 BEGIN 16:11:30.449950000
From START worker: task 1 ENDED 16:11:30.449960000
From START worker: task 2 BEGIN 16:11:30.449964000
From START worker: task 2 ENDED 16:11:30.449967000
From START worker: task 3 BEGIN 16:11:30.449969000
From START worker: task 3 ENDED 16:11:30.449972000
From START worker: task 4 BEGIN 16:11:30.449976000
From MIDDLE worker: task 0 BEGIN 16:11:30.449950000
From START worker: task 4 ENDED 16:11:30.449979000
From START worker: task 5 BEGIN 16:11:30.449991000
From START worker: task 5 ENDED 16:11:30.449994000
From START worker: task 6 BEGIN 16:11:30.449997000
From START worker: task 6 ENDED 16:11:30.450000000
From MIDDLE worker: task 0 ENDED 16:11:30.450024000
From MIDDLE worker: task 1 BEGIN 16:11:30.450041000
From END worker: task 0 BEGIN 16:11:30.450049000
From END worker: task 0 ENDED 16:11:30.450076000
From MIDDLE worker: task 1 ENDED 16:11:30.450079000
From MIDDLE worker: task 2 BEGIN 16:11:30.450093000
From END worker: task 1 BEGIN 16:11:30.450102000
From END worker: task 1 ENDED 16:11:30.450123000
From MIDDLE worker: task 2 ENDED 16:11:30.450125000
From MIDDLE worker: task 3 BEGIN 16:11:30.450136000
From END worker: task 2 BEGIN 16:11:30.450143000
From END worker: task 2 ENDED 16:11:30.450159000
From MIDDLE worker: task 3 ENDED 16:11:30.450171000
From MIDDLE worker: task 4 BEGIN 16:11:30.450185000
From END worker: task 3 BEGIN 16:11:30.450193000
From MIDDLE worker: task 4 ENDED 16:11:30.450207000
From END worker: task 3 ENDED 16:11:30.450212000
From MIDDLE worker: task 5 BEGIN 16:11:30.450216000
From END worker: task 4 BEGIN 16:11:30.450228000
From END worker: task 4 ENDED 16:11:30.450240000
From MIDDLE worker: task 5 ENDED 16:11:30.450251000
From MIDDLE worker: task 6 BEGIN 16:11:30.450264000
From END worker: task 5 BEGIN 16:11:30.450272000
From END worker: task 5 ENDED 16:11:30.450288000
From MIDDLE worker: task 6 ENDED 16:11:30.450296000
From END worker: task 6 BEGIN 16:11:30.450313000
From END worker: task 6 ENDED 16:11:30.450332000
Press <RETURN> to close this window...
```

Рис. 4.1: Пример работы ПО.

### 4.3 Время выполнения алгоритмов

Алгоритм тестировался на данных, сгенерированных случайным образом.

В таблице 4.1 предоставлено время работы над каждым отрезком в предоставленном примере каждого из выделенных этапов.

Из таблицы видно, что среднее время выполнения этапа 1 составляет  $\approx 17428.6$  наносекунд. Среднее время выполнения этапа 2 составляет  $\approx 38285.7$  наносекунд. Среднее время выполнения этапа 3 составляет  $\approx 18571.4$  наносекунд. Таким образом, этап 1 сравним по среднему времени выполнения с этапом 2. Но после выполнения этапа 1 заметно, что последующие вызовы функции работают за константное время, равное 3000 наносекунд, что при наличии начальных "прогревочных" запусков вылилось бы в факт того, что этап 1 не был бы сопоставим по среднему времени выполнения с этапом 3. Этап 2 является самым долго выполняющимся.

Таблица 4.1: Замеры времени для выполнения выделенных этапов.

Номер отрезка	Время обработки, нс		
	Этап 1	Этап 2	Этап 3
0	97000	74000	27000
1	10000	38000	21000
2	3000	32000	16000
3	3000	35000	19000
4	3000	22000	12000
5	3000	35000	16000
6	3000	32000	19000

## Вывод

При сравнении результатов замеров по времени стало известно, что самым быстрым этапом конвейера оказался этап 1. При этом, самым медленным из трех рассмотренных - этап 2.

В среднем этап 1 работает быстрее этапа 2 на  $\approx 20857.1$  наносекунд. При этом, при четвертой обработке отрезка разница в скорости выполнения составила 32000 наносекунд.

Этап 3 в среднем работает быстрее этапа 2 на  $\approx 19714.3$  наносекунд. При этом, при шестой обработке отрезка разница в скорости выполнения составила 19000 наносекунд.

Таким образом, среднее время выполнения алгоритма для каждого отрезка составило  $\approx 74285.71$  наносекунд.

# Заключение

В ходе выполнения лабораторной работы была выполнена цель и следующие задачи:

- 1) было изучено асинхронное взаимодействие на примере конвейерной обработки данных;
- 2) была спроектирована система конвейерных вычислений;
- 3) была реализована система конвейерных вычислений;
- 4) была протестирована реализованная система;
- 5) был подготовлен отчёт по проведенной работе.

Исследования показали, что в среднем:

- 1) этап 1 работает быстрее этапа 2 на  $\approx 20857.1$  наносекунд;
- 2) этап 3 работает быстрее этапа 2 на  $\approx 19714.3$  наносекунд;
- 3) среднее время выполнения алгоритма составило  $\approx 74285.71$  наносекунд.

# Литература

- [1] Perl. Примеры программ [Электронный ресурс]. Режим доступа: <http://mech.math.msu.su/~shvetz/54/inf/perl-problems/> (дата обращения 09.12.2020).
- [2] Решение задачи коммивояжера для поиска оптимального плана перевозок предприятия (на примере ООО «Фабрика еды») [Электронный ресурс]. Режим доступа: <https://perm.hse.ru/data/2015/02/16/1092093493/%D0%A0%D0%B5%D1%88%D0%B5%D0%BD%D0%B8%D0%B5%20%D0%B7%D0%B0%D0%B4%D0%B0%D1%87%D0%B8%20%D0%BA%D0%BE%D0%BC%D0%BC%D0%B8%D0%B2%D0%BE%D1%8F%D0%B6%D0%B5%D1%80%D0%B0.pdf> (дата обращения 09.12.2020).
- [3] М.В. Ульянов. Ресурсно-эффективные компьютерные алгоритмы. ФИЗМАТЛИТ, 2008. с. 304.
- [4] C++ standart [Электронный ресурс]. Режим доступа: <http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2013/n3690.pdf> (дата обращения 01.12.2020).
- [5] Qt documentation [Электронный ресурс]. Режим доступа: <https://doc.qt.io/> (дата обращения 01.12.2020).
- [6] Boost library [Электронный ресурс]. Режим доступа: <https://www.boost.org/> (дата обращения 09.12.2020).