Санкт-Петербургский государственный университет

Математическое обеспечение и администрирование информационных систем
Кафедра системного программирования

Иванова Марина Андреевна

# Реализация построения разбиения Вороного для ОСРВ МАКС

Выпускная квалификационная работа бакалавра

Научный руководитель:
к. т. н., доц. кафедры СП Литвинов Ю. В.

Научный консультант:
заместитель директора по направлению разработки системного ПО
ООО «АстроСофт» Бойко П. В.

Рецензент:
ведущий программист-математик
ООО «АстроСофт» Бабин Г. В.

Санкт-Петербург
2018

SAINT-PETERSBURG STATE UNIVERSITY

Software and Administration of Information Systems
Software Engineering

Marina Ivanova

# Implementation of the construction of Voronoi partitioning for RTOS MAKS

Graduation Thesis

Scientific supervisor:
Candidate of Engineering Sciences Yurii Litvinov

Consultant:
Deputy director OOO "AstroSoft"
Pavel Boiko

Reviewer:
Mathematical programmer OOO "AstroSoft"
German Babin

Saint-Petersburg
2018

# Оглавление

# Введение

Развитие современных технологий приводит к все большей автоматизации процессов, выполняемых человеком в различных сферах жизнедеятельности, таких как медицина, образование, сельскохозяйственное, промышленное и военное дело. В настоящее время имеется множество устройств, способных заменить людей в опасных или труднодостижимых для человека местах, ускорить и облегчить выполнение различных задач.

На данный момент актуальной темой является создание автоматизированных технологических систем, реализующих групповое взаимодействие агентов для выполнения общей задачи. Например, построение карты местности, поисково-спасательные операции, орошение полей, тушение пожара, обеспечение эффективного слежения группой беспилотников за стратегически важными объектами, и т.д. Такие задачи могут быть сведены к задаче о покрытии поля. Поле — это плоскость с заданной декартовой системой координат, состоящая из клеток, каждая из которых представляет собой одну квадратную единицу. В задаче о покрытии поля группа агентов должна полностью его покрыть, а именно посетить все клетки поля.

Для эффективного выполнения задачи покрытия поля несколькими агентами необходимо разделить между ними поле, т.к. оно может оказаться существенно больше, чем количество агентов. Такое разбиение можно построить с помощью разбиения Вороного, или диаграммы Вороного [9]. Достоинством метода разбиения Вороного является учет начального положения агентов (т.к агенты являются точками, для которых строится диаграмма Вороного), получение непересекающихся областей, представляющих собой выпуклые многоугольники, а также возможность реализации данного метода разного рода модификациями алгоритмов с различными требованиями к ресурсам. Таким образом, для решения задачи о покрытии поля, мы можем поручить построение разбиения любому из агентов, сделав полученное разбиение доступным

оставшимся участникам.

Для обмена информацией между агентами и обеспечения ее согласованности удобно использовать операционную систему реального времени (ОСРВ) с поддержкой распределенной памяти. В данной работе используется ОСРВ МАКС [15], которая может быть применена для создания систем, включающих несколько интеллектуальных узлов — аппаратных средств, общающихся между собой. ОСРВ МАКС предназначена для систем реального времени и других встраиваемых систем под управлением ARM микроконтроллеров на базе 32-битных ядер Cortex-M3, Cortex-M4, Cortex-M4F с RISC архитектурой. В данной ОСРВ предложена модель общей распределенной памяти (distributed shared memory, DSM) — DSM МАКС [13], которая обеспечивает каждого агента системы доступом к общей памяти, возможностью писать и читать данные.

Таким образом, данная работа посвящена реализации разбиения поля с помощью разбиения Вороного для ОСРВ МАКС.

# 1. 1. Постановка задачи

Целью данной работы является решение задачи о разделении поля методом разбиения Вороного для нескольких агентов, использующих ОСРВ МАКС. Для достижения этой цели были поставлены следующие задачи.

1. Изучение алгоритмов решения задачи разбиения Вороного.
2. Выбор алгоритмов, решающих задачу о разбиении и подходящих для работы на микроконтроллерах.
3. Реализация выбранных алгоритмов или добавление необходимой функциональности к уже существующим реализациям (если такие есть).
4. Проведение апробации алгоритмов разбиения Вороного в ОСРВ МАКС с оценкой эффективности.

# 2. Обзор существующих алгоритмов разбиения Вороного

Разбиение Вороного для заданного набора точек на плоскости — это разбиение плоскости на ячейки Вороного, соответствующие конкретной точке данного набора. Ячейка Вороного — это геометрическое место точек плоскости, наиболее близких к точке из данного набора [5].

## 2.1. Инкрементальный алгоритм

Диаграмма строится инкрементально. Предположим, что мы уже имеем диаграмму Вороного, построенную для $N$ точек. Для добавления точки $P_{N+1}$ в диаграмму нам необходимо выполнить следующие шаги, изображенные на (рис. 1).

- Среди ячеек Вороного, которые мы уже построили, найти ту, в которую входит точка $P_{N+1}$. Пусть эта ячейка соответствует точке $P_i$.

- Провести серединный перпендикуляр для точек $P_{N+1}$ и $P_i$. Построенный серединный перпендикуляр пересечет границу ячейки, соответствующую точке $P_i$, которая также является границей другой ячейки, соответствующей, например, точке $P_k$. Далее будем строить серединный перпендикуляр между $P_{N+1}$ и $P_k$ и т.д.

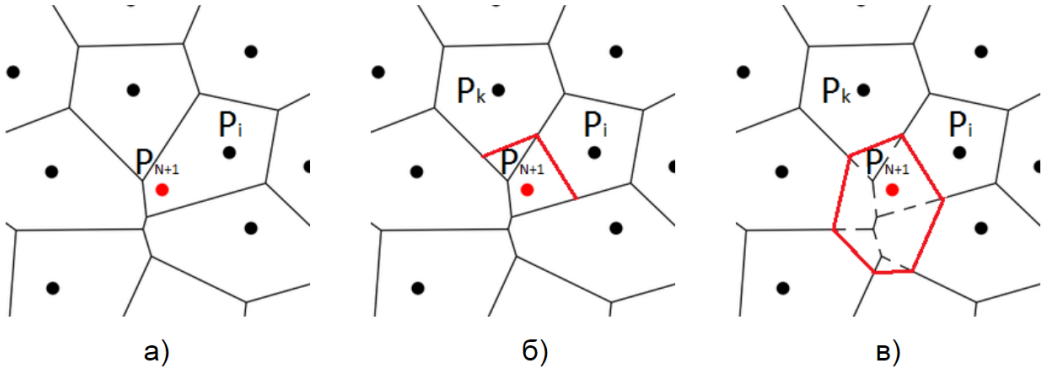


Рис. 1: Построение диаграммы Вороного инкрементальным алгоритмом: (а) первый шаг алгоритма; (б) второй шаг алгоритма; (в) последний шаг алгоритма.

## 2.2. «Разделяй и властвуй»

Пусть у нас есть исходное множество точек $P$. Данный алгоритм работает следующим образом. Множество P делится на два примерно равных подмножества $P_1$ и $P_2$. Для каждого из этих подмножеств рекурсивно строится диаграмма Вороного. Затем происходит восстановление диаграммы для исходного множества слиянием полученных диаграмм. Для того, чтобы можно было реализовать слияние двух диаграмм Вороного, множества, для которых строились эти диаграммы, должны быть разделяемыми (т.е. должна быть возможность определения, к какому из множеств принадлежит каждая точка из $P$), что мы можем обеспечить на этапе разделения исходного множества $P$, например, просто отсортировав все точки по ординате. Тогда по теореме из [3] следует, что диаграмма Вороного для исходного множества будет равна $Vor(P_1) \cap \pi_l \cup Vor(P_2) \cap \pi_r$ , где $Vor(P_1)$, $Vor(P_2)$ — диаграммы Вороного для $P_1$ и $P_2$ соответственно, а $\pi_l$, $\pi_r$— левая часть диаграммы Вороного $P_1$ и правая часть диаграммы Вороного $P_2$ относительно разделяющей линии.

Такой алгоритм имеет асимптотическую сложность порядка $O(NlogN)$, где $N$ — количество точек в исходном множестве. К тому же данный алгоритм можно распараллелить, что является его привлека-

тельной чертой, но он вызывает трудности при численной реализации.

## 2.3. Алгоритм Форчуна

При реализации данного алгоритма диаграмма Вороного представляется в виде графа. Идея алгоритма следующая: имеется набор из $N$ точек на плоскости и некоторая заметающая прямая — прямая, которая движется, например, сверху вниз. Точки из исходного набора при таком движении заметающей прямой в какие-то моменты могут оказаться лежащими на этой прямой. В таком случае строится парабола, для которой вводятся контрольные точки — точки пересечения данной параболы с уже существующими параболами. Затем вместе с движением заметающей прямой парабола расширяется и контрольные точки сдвигаются. Кривая, образованная дугами, заключенными между контрольными точками построенных парабол, называется «береговой линией».

Введем понятия «событие-точка» и «событие-круг».

«Событие-точка» возникает при движении заметающей прямой, когда какая-то точка из набора оказывается принадлежащей этой прямой. Когда происходит событие этого типа, необходимо построить новую параболу, фокусом которой является точка из набора, которая оказалась на заметающей прямой, а директрисой — заметающая прямая. В момент, когда заметающая прямая проходит через точку множества, парабола, соответствующая этой точке и заметающей прямой, представляет собой луч, выходящий из этой точки и направленный в сторону «береговой линии». При дальнейшем движении заметающей прямой ветви параболы расширяются и появляется пара контрольных точек. Во время формирования «события-точки» необходимо перестроить «береговую линию», добавив туда дугу, образованную параболой, построенной для новой точки. Поэтому для данной точки необходимо определить соответствующую дугу «береговой линии», которая затем будет разделена на две путем вставки бесконечно малой дуги в точ-

ку пересечения луча, исходящего из точки, и «береговой линии». При дальнейшем движении заметающей прямой эта дуга будет расширяться до тех пор, пока не соединится с другими ребрами диаграммы. На (рис. 2) показано движение заметающей прямой, следствием которого является возникновение «события-точки».

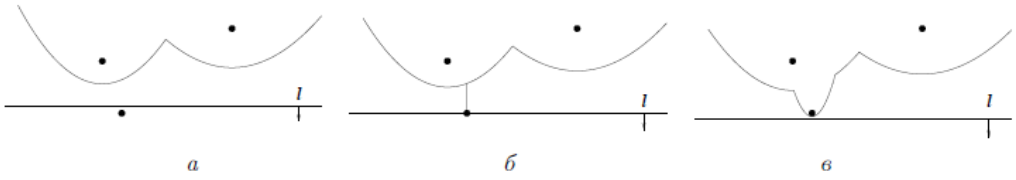


Рис. 2: Возникновение «события-точки»: (а) прямая l не достигла точки исходного набора; (б) прямая l достигла точки исходного набора, дуга «береговой линии» разделилась на две части бесконечно малой дугой; (в) прямая l пересекает точку исходного набора, дуга на «береговой линии» расширяется, [9].

«Событие-круг» — это возникновение новой ячейки Вороного. Событие такого типа создается при «схлопывании» дуг «береговой линии», что генерируется динамически в процессе выполнения работы алгоритма. При формировании данного события возникает удаление дуги из «береговой линии», процесс которого изображен на (рис. 3).

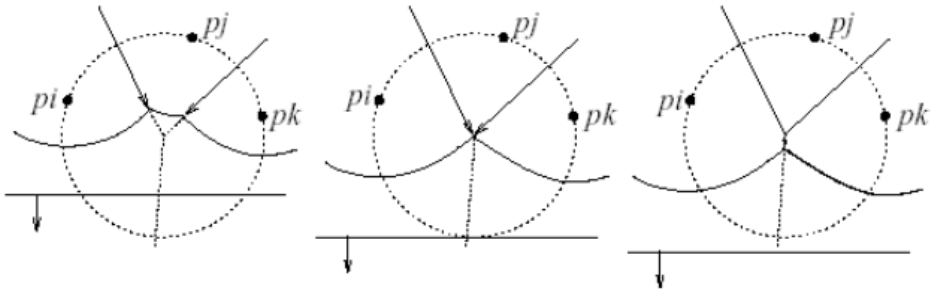


Рис. 3: Возникновение «события-круг», [9].

В [14] представлена теорема о том, что каждая вершина диаграммы Вороного есть в точности точка пересечения трех ребер диаграммы. Из этой теоремы выведено следствие о том, что вершинами диаграммы

Вороного являются центры окружностей, определенных тремя точками исходного множества и расстоянием до заметающей прямой должен быть радиус этой окружности. При этом в такой окружности нет никаких точек из исходного множества. Таким образом, в момент удаления дуги из «береговой линии» возникает столкновение двух контрольных точек и происходит соединение двух ребер диаграммы. В [3] доказано, что удаление дуги из «береговой линии» может происходить только при «событии-круг».

Такой алгоритм имеет асимптотическую сложность $O(NlogN)$, где $N$ — количество точек в исходном множестве. Данный алгоритм является более простым для понимания и реализации, чем алгоритм «Разделяй и властвуй».

## 2.4. Сравнение алгоритмов и существующих библиотек

Для решения задачи построения Вороного было решено не использовать алгоритм «Разделяй и властвуй», т.к рекурсивные функции занимают стек на адрес возврата и стековые кадры — чем больше локальных переменных, тем больше стека расходуется при вызове рекурсивной функции, что плохо при реализации в ОСРВ МАКС. Для построения разбиения Вороного алгоритмом Форчуна существует несколько библиотек.

- SplashGeom [7], в данной библиотеке нет поддержки случаев, когда все точки исходного множества лежат на одной окружности.
- LEDA [4], в данной библиотеке есть реализация алгоритма Форчуна и преобразование диаграммы Вороного в триангуляцию Делоне и обратно.
- CGAL [2] может строить диаграмму Вороного для точек и сегментов, но не может проводить вычисления с заданной точностью.

- S-Hull [6] используется для построения триангуляции Делоне для набора точек, сами разработчики утверждают, что она является ненадежной реализацией.

- Boost.Polygon Voronoi [1] отличается от CGAL тем, что поддерживает вычисления в пределах фиксированной точности. В данной библиотеке нет реализации разбиения Вороного в ограничивающей рамке, т.е остается часть поля, не разбитая между самыми «внешними» точками.

В качестве решения задачи построения разбиения Вороного алгоритмом Форчуна была выбрана библиотека Boost.Polygon Voronoi. Данная библиотека позволяет получить вершины и ребра диаграммы Вороного.

Существующих библиотек, позволяющих реализовать построение диаграммы Вороного с помощью инкрементального алгоритма не найдено. Поэтому возникает необходимость его реализации.

# 3. Реализация алгоритма Форчуна

В качестве решения построения диаграммы Вороного была выбрана библиотека Boost.Polygon Voronoi. Данная библиотека позволяет получить вершины и ребра диаграммы Вороного, но не позволяет ограничить поле некой рамкой. В связи с этим возникают ребра, имеющие одну «хорошую» вершину — образованную «событием-круг», и одну бесконечную, т.к одна ветвь параболы участвует в «событии-круг», а вторая — нет. Поэтому возникла необходимость реализовать недостающую функциональность, а именно отобразить бесконечные вершины в конечные точки.

Для решения данной задачи рассматривалось несколько решений.

Первое: через все точки исходного набора, для которых ячейка Вороного содержит бесконечную вершину Вороного, построить многоугольник (так, чтобы он содержал все такие точки исходного набора) и затем провести прямые через вершину ребра Вороного, содержащую бесконечную вершину, и середину соответствующей стороны многоугольника.

Этот способ рассматривался, т.к. каждая такая прямая будет равноудалена от точек исходного набора. По (рис. 4) видно, что если $A$ и $B$ есть точки исходного набора, а $C$ — вершина диаграммы, то прямая, содержащая отрезок $CM$ будет равноудалена от точек $A$ и $B$. По построению $\angle AMA' = \angle BMB'$, $BM = AM$, а значит по теореме о равенстве прямоугольных треугольников $\triangle AMA' = \triangle BMB'$. Откуда следует, что $BB' = AA'$. И значит полученные ячейки будут удовлетворять условию диаграммы Вороного равноудаленности точек от заданной.

Но данное решение было отклонено, т.к, во-первых, построенный многоугольник не всегда будет выпуклым, а значит невозможно однозначно построить такой многоугольник. Во-вторых, стороны полученного многоугольника могут пересекать уже полученные ячейки Вороного, тогда ячейки Вороного могут получиться невыпуклыми многоугольниками, а диаграмма Вороного состоит из выпуклых.

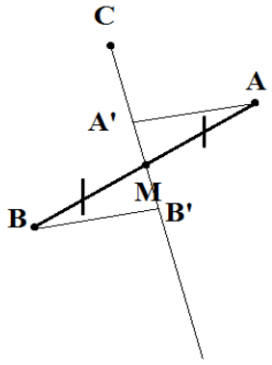


Рис. 4: $AA'$ и $BB'$ — перпендикуляры к прямой $CM$, $M$ — середина отрезка $AB$.

Второе: т.к в библиотеке Boost.Polygon реализован алгоритм Форчуна, было решено восстановить «боковые» параболы — те, которые имеют одну ветвь не участвующую в «событии-круг».

На основе определения параболы, вершины и фокуса, мы можем восстановить параболу, а именно, для каждой точки исходного набора, которой соответствует ячейка Вороного, содержащая бесконечную вершину, будем строить параболу, считая за фокус эту исходную точку. Для бесконечной вершины мы знаем второй конец ребра Вороного и по действию алгоритма Форчуна этот конец является контрольной точкой, а значит, лежит на параболе, которую мы хотим восстановить. Восстановленную таким способом параболу показывает (рис. 5).

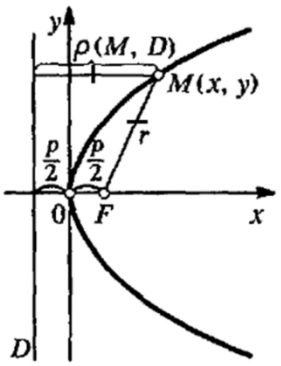


Рис. 5: После вычисления диаграммы Вороного, с помощью библиотеки Boost.Polygon Voronoi мы знаем точку $F(x1, y1)$ и точку $M(x2, y2)$, [8]

Для удобства рассмотрим случай, когда вершина параболы лежит

в точке $(0, 0)$.

С помощью формулы

$$dist(F, M) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{1}$$

вычислим расстояние между фокусом и точкой, которая лежит на требуемой параболе. По определению параболы и формулы (1) мы знаем расстояние от точки параболы до директрисы параболы. Сделав поправку

$$dist(F, M) - (x_2 - x_1) \tag{2}$$

мы найдем расстояние $p$.

Тогда мы можем использовать каноническое уравнение параболы

$$y^2 = 2 \cdot p \cdot x \tag{3}$$

с помощью которого найдем точку пересечение с рамкой поля и заменим бесконечную вершину на эту точку.

Данные действия проделаем со всеми точками исходного набора, для которых есть бесконечные вершины, и таким образом достроим диаграмму Вороного, как показано на (рис.6).

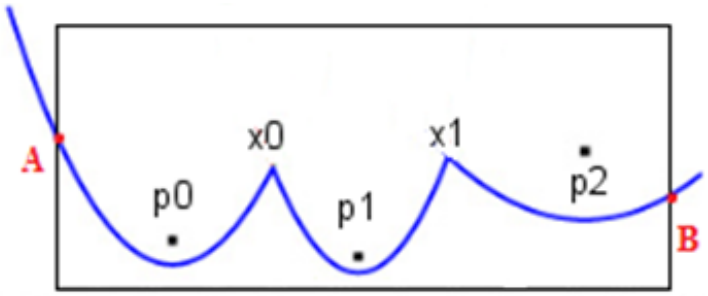


Рис. 6: На рисунке $x_0$ и $x_1$ — вершины Вороного, являющиеся конечными концами соответствующих ребер, которые имеют бесконечные вершины. $A$ и $B$ — точки, на которые заменяются бесконечные вершины.

Такое решение — восстановление параболы, полученной в алгоритме

Форчуна — является последним шагом в данном алгоритме, а значит будет удовлетворять условиям диаграммы Вороного.

# 4. Реализация инкрементального алгоритма

При реализации данного алгоритма диаграмму Вороного предлагается хранить в структуре данных DCEL (англ. doubly connected edge list — реберный список с двойными связями) [16]. Такая структура состоит из трёх компонент.

- Вершина — содержит координаты точки и ссылку на половинное ребро, исходящее из этой вершины.
- Поверхность — содержит список указателей на ребра, образующие ее границу.
- Половинное ребро — содержит указатель на точку исхода, указатель на инцидентную поверхность, указатели на предыдущее и следующее ребро.

Как было описано ранее, для того, чтобы построить новую ячейку Вороного, необходимо найти ячейку среди существующих, в которую попадает новая точка. Т.к используется структура данных DCEL и для каждой исходной точки мы имеем ссылку на ее поверхность, то мы легко получаем многоугольник, образующий ячейку Вороного. После этого вычисляются все «соседние» ячейки, те, границы которых являются общими с найденной поверхностью. И, затем, производится построение серединных перпендикуляров для новой точки и точек, являющихся точками «соседних» ячеек. Такие серединные перпендикуляры пересекают полученные ранее ячейки Вороного, тем самым образуя новую ячейку Вороного. А именно, ребра «соседних» ячеек перестраиваются, меняя свой конец на точку пересечения перпендикуляра и ячейки, и создаются новые ребра для новой ячейки Вороного. При этом дуги, которые находятся внутри полученной ячейки Вороного, должны быть удалены.

При добавлении новой точки в ячейки, имеющие ребра, являющиеся гранями рамки, возникает проблема с тем, что новая ячейка Вороного может образовать ломаную, не являющуюся многоугольником. Поэтому было решено после нахождения «соседей», если они не образуют «кольцо» вокруг нашей точки, расположить их в виде цепочки, начиная с первой «соседней» ячейки против часовой стрелки. В таком случае, после построения ломаной, мы либо соединим начало и конец, либо добавим ребра, образованные гранями рамки.

С использованием DCEL данный алгоритм будет иметь асимптотическую сложность порядка $O(N^2)$, где $N$ — количество точек в исходном множестве.

# 5. Эксперименты

## 5.1. Апробация на ОСРВ МАКС

Решение задачи о разделении поля предполагается в дальнейшем использовать в задаче о покрытии поля несколькими агентами, снабженными ОСРВ МАКС с общей распределенной памятью МАКС DSM. Поэтому рассмотрим особенности данной операционной системы.

ОСРВ МАКС в текущей реализации предназначена для работы на недорогих микроконтроллерах. Объем внутреннего ОЗУ в таких контроллерах обычно невелик, а исполнение программ во внешнем ОЗУ медленнее, чем во встроенной флэш-памяти [12]. Поэтому в данной ОСРВ:

- программа пользователя и сама ОС компилируются и компонуются монолитно. И они обе находятся во флэш-памяти, в связи с тем, что на некоторых контроллерах исполнение программы в ОЗУ снижает производительность, а в некоторых случаях — просто невозможно (если в системе нет внешней микросхемы ОЗУ);

- в ОСРВ МАКС нет понятия «процесс», и изолированные процессы в пределах одного микроконтроллера запустить невозможно. В данной ОСРВ существует понятие «задача приложения пользователя» и именно с ними работает микроядро, выполняет их планирование и взаимодействие [11].

Еще одной отличительной чертой данной ОСРВ является то, что в ней реализован подход ООП. И для того, чтобы можно было выполнить пользовательскую программу, необходимо в главном методе (точке входа приложения) создать экземпляр наследника класса Application. Для наследника необходимо реализовать виртуальный метод Initialize(), где инициализируется приложение. После чего можно создать задачу — экземпляр наследника класса Task. Для наследника класса Task нужно

реализовать виртуальный метод Execute(). После этого можно добавить задачу в планировщик с помощью метода Add(). При добавлении задачи есть возможность явного задания размера стека, необходимого для выполнения задачи. Его максимальным размером является 3760 32-битных слов. Именно из-за этого ограничения для решения задачи о разбиении поля пришлось отказаться от использования алгоритма «Разделяй и властвуй», т.к рекурсивные функции тратят стек не только на адреса возврата, но и на стековые кадры, и значит, больше стека используется при рекурсивных вызовах, а этого лучше избегать при работе с ограниченной памятью (иначе мы должны знать, что цепочки вызовов никогда не станут слишком длинными) [11].

Также стоит учесть тот факт, что разработчики ОСРВ МАКС не рекомендуют использовать динамическое выделение памяти, из-за того, что операцию выделения памяти нельзя отнести к операциям реального времени т.к ее быстродействие невозможно предсказать, а системы реального времени подразумевают гарантированное быстродействие [17]. Поэтому при реализации алгоритмов по возможности использовалось статическое выделение памяти и память на куче выделялась на этапе инициализации программы.

## 5.2. Оценка эффективности работы алгоритмов в ОСРВ МАКС

Для данной реализации была проведена серия тестов. Каждая серия проводилась следующим образом. Было взято поле размера 10x10 и программа запускалась несколько раз на одном наборе из $M$ случайных точек этого поля. Для каждого теста по наблюдаемым данным было посчитано математическое ожидание и среднее квадратическое отклонение. Расчеты проводились по формулам:

$$E = \frac{1}{n} \cdot \sum_{i=1}^{n} X_i \tag{4}$$

$$S = \sqrt{\frac{1}{n-1} \cdot \sum_{i=1}^{n} (X_i - E)^2} \tag{5}$$

где $E$ — математическое ожидание, $S$ — среднее квадратическое отклонение, $X_i$ — времена, полученные при проведении тестов, $n$ — количество проведенных тестов в серии. Такой метод оценки описан в [18].

Тесты проводились на системе:

- ОСРВ МАКС
- Cortex-M4 микроконтроллер серии STM32 F4:
  - ARM 32-bit Cortex-M4 CPU
  - 180 МГц
  - 256 Кб RAM
  - 2 Мб ROM

Ниже представлена таблица, содержащая результаты расчета выборочной несмещенной дисперсии.

Таблица 1: среднее квадратическое отклонение результата работы программы в различных сериях тестов.

| серия теста | количество точек | Boost.Polygon Voronoi | | Инкрементальный алгоритм DCEL | |
|---|---|---|---|---|---|
| | | мат. ожидание, (мс) | среднее квадратическое отклонение, (мс) | мат. ожидание, (мс) | среднее квадратическое отклонение, (мс) |
| 1 | 3 | 2.09 | 0.30 | 1.90 | 0.30 |
| 2 | 5 | 6.30 | 0.75 | 4.80 | 0.68 |
| 3 | 7 | 13.71 | 1.31 | 10.60 | 1.11 |
| 4 | 9 | 24.41 | 1.75 | 15.60 | 1.56 |
| 5 | 11 | 38.58 | 2.83 | 21.20 | 2.31 |

Таким образом, мы видим, что время работы реализации инкрементного алгоритма меньше времени работы реализации алгоритма Форчуна в библиотеке Boost.Polygon Voronoi.

# Заключение

Таким образом, в рамках данной работы получены следующие результаты.

1. Изучены алгоритмы Форчуна, Инкрементальный и «Разделяй и властвуй», позволяющие реализовать разбиение поля между объектами. Рассмотрены библиотеки, реализующие разбиение Вороного алгоритмом Форчуна.

2. Реализован алгоритм Форчуна при помощи библиотеки Boost.Polygon Voronoi. Разработан метод, позволяющий по построенной диаграмме, достроить разбиение Вороного полностью.

3. Реализован инкрементальный алгоритм с использованием структуры данных DCEL.

4. Произведены апробация в ОСРВ МАКС и сравнение работы алгоритмов Форчуна и инкрементального. В ходе сравнения было выявлено, что реализованный инкрементальный алгоритм со структурой данных DCEL по времени работает быстрее, чем дополненный функциональностью алгоритм Форчуна библиотеки Boost.Polygon Voronoi.

5. Тезисы доклада с основными результатами работы опубликованы в сборнике конференции "Современные технологии в теории и практике программирования" [10].

6. Представлен доклад на конференции SYRCoSE-2018.

7. Программная разработка расположена на веб-сервисе для хостинга IT-проектов GitHub https://github.com/ivm23/Voronoi.

# Список литературы


[1] Boost.Polygon Voronoi. — URL: http://www.boost.org/doc/libs/1_61_0/libs/polygon/doc/voronoi_main.htm (online; accessed: 06.12.2017).

[2] CGAL. — URL: https://www.cgal.org/ (online; accessed: 06.12.2017).

[3] De Berg M. Van Kreveld M. Computational Geometry. Algorithms and Applications. Second, Revised Edition. Berlin: Springer-Verlag. — 2000. — P. 367.

[4] LEDA. — URL: http://www.algorithmic-solutions.com/leda/index.htm (online; accessed: 06.12.2017).

[5] Malinauskas K. K. Dynamic construction of abstract Voronoi diagrams, Fundamentalnayai prikladnaya matematika, vol. 13. — 2007. — P. 133–146.

[6] S-HULL. — URL: http://www.s-hull.org/ (online; accessed: 06.12.2017).

[7] SplashGeom. — URL: https://github.com/izaharkin/SplashGeom (online; accessed: 06.12.2017).

[8] Б.П. Ефимов А.В. Демидович. Сборник задач по математике. Berlin: Springer-Verlag. — 2001. — P. 288.

[9] Диаграмма Вороного и её применения. — URL: https://habrahabr.ru/post/309252/ (дата обращения: 06.12.2017).

[10] Иванова М.А. Литвинов Ю.В. Решение задачи разбиения поля методом диаграммы Вороного // Современные технологии в теории и практике программирования, сборник материалов конференции. СПб.: Изд-во Политехн. ун-та. — 2018. — С. 81–84.

[11] ОПЕРАЦИОННАЯ СИСТЕМА РЕАЛЬНОГО ВРЕМЕНИ ДЛЯ МУЛЬТИАГЕНТНЫХ КОГЕРЕНТНЫХ СИСТЕМ. Руководство программиста. — 2018. — P. 56.

[12] Обзор одной российской RTOS, часть 4. Полезная теория / Хабр. — URL: https://habr.com/post/337476/ (online; accessed: 21.05.2018).

[13] П.В. Бойко. Распределенная общая память как способ организации взаимодействия в мультиагентных системах // Научно-аналитический журнал. Инновации и инфестиции. — 2017. — С. 113–116.

[14] Препарата Ф. Шеймос М. Вычислительная геометрия: Введение. — 1989. — P. 478.

[15] Российская операционная система реального времени для IT-оборудования и Интернета вещей. — URL: https://www.astrosoft.ru/products/development/rtos-macs/ (online; accessed: 15.04.2018).

[16] С.Н. Карабцев С. В. Стуколов. Построение диаграммы Вороного и определение границ области в методе естественных соседей. Вычислительные технологии Том 13, № 3. — 2008. — P. 65–80.

[17] Страуструп Бьерн. Программирование. Принципы и практика использования C++. — 2011. — P. 1248.

[18] Э.В. Егорова. Математика и информатика. Учебное пособие по всему курсу. — 2008. — P. 94.