Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и кибербезопасности

**Высшая школа кибербезопасности**

**КУРСОВАЯ РАБОТА**

**Оптимизация моделирования решения Кубика Рубика 3x3**

по дисциплине «Языки программирования»

Выполнили

студенты гр. 5151003/30002 И. А. Штарев

Д. А. Плотников

Руководитель

асс. преподавателя Е. М. Орел

«DD» MM YYYY г.

Санкт-Петербург

YYYY

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc184236282)

[1. ТЕОРЕТИЧЕСКИЕ ИССЛЕДОВАНИЯ 4](#_Toc184236283)

[1.1. ВОЗМОЖНОСТИ ДЛЯ МАШИННО-НЕЗАВИСИМОЙ ОПТИМИЗАЦИИ 4](#_Toc184236284)

[1.2. ВОЗМОЖНОСТИ ДЛЯ АЛГОРИТМИЧЕСКОЙ ОПТИМИЗАЦИИ 5](#_Toc184236285)

[2. МАШИННО-НЕЗАВИСИМАЯ ОПТИМИЗАЦИЯ 6](#_Toc184236286)

[2.1. ОПТИМИЗАЦИЯ БУФЕРОВ 6](#_Toc184236287)

[2.2. ОПТИМИЗАЦИЯ 2 7](#_Toc184236288)

[3. АЛГОРИТМИЧЕСКАЯ ОПТИМИЗАЦИЯ 8](#_Toc184236289)

[3.1. ТЕКСТ 8](#_Toc184236290)

[3.2. ТЕКСТ 8](#_Toc184236291)

[4. АНАЛИЗ ОПТИМИЗАЦИЙ 9](#_Toc184236292)

[ЗАКЛЮЧЕНИЕ 10](#_Toc184236293)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 11](#_Toc184236294)

[ПРИЛОЖЕНИЕ А. БЛОК-СХЕМЫ ФУНКЦИЙ ОТРИСОВКИ ОКОН 13](#_Toc184236295)

[ПРИЛОЖЕНИЕ Б. ЛИСТИНГИ ФУНКЦИЙ ОТРИСОВКИ ОКОН 14](#_Toc184236296)

[ПРИЛОЖЕНИЕ В. БЛОК-СХЕМЫ КЛЮЧЕВЫХ ФУНКЦИЙ АЛГОРИТМА РЕШЕНИЯ КУБИКА 16](#_Toc184236297)

[ПРИЛОЖЕНИЕ Г. ЛИСТИНГИ КЛЮЧЕВЫХ ФУНКЦИЙ АЛГОРИТМА РЕШЕНИЯ КУБИКА 17](#_Toc184236298)

# ВВЕДЕНИЕ

В современных вычислительных системах важно учитывать как эффективность алгоритмов, так и производительность программного обеспечения. Одной из задач, которая требует оптимизации, является решение кубика Рубика с использованием графических библиотек и алгоритмов для его моделирования. В предыдущей курсовой работе была реализована программа для решения кубика Рубика с использованием языка C и библиотеки OpenGL с GLFW. Однако, несмотря на свою функциональность, программа имеет возможности для улучшения как с точки зрения алгоритмической, так и машинной оптимизации, что обеспечит ускорение выполнения и улучшение пользовательского опыта.

Целью данной курсовой работы является оптимизация исходной программы, созданной в рамках предыдущей работы. В процессе оптимизации будет проведено профилирование программы с целью выявления узких мест и неэффективных участков кода. На основе полученных данных будут предложены и реализованы как алгоритмические, так и машинно-независимые оптимизации, что позволит снизить вычислительные затраты и улучшить общую производительность. Важным аспектом будет также оценка изменения алгоритмической сложности программы и сравнение результатов до и после оптимизаций.

В рамках выполнения работы будет проведен анализ исходной программы с применением методов профилирования и оценки сложности алгоритмов. Будет предложено и реализовано несколько вариантов оптимизаций, среди которых особое внимание будет уделено алгоритмическим улучшениям и кодовым оптимизациям, не зависящим от конкретной архитектуры. Программная система будет проанализирована с точки зрения полученных результатов, а также приведены графики, на которых будут отражены изменения в производительности программы после каждого этапа оптимизации.

# 1. ТЕОРЕТИЧЕСКИЕ ИССЛЕДОВАНИЯ

# 1.1. ВОЗМОЖНОСТИ ДЛЯ МАШИННО-НЕЗАВИСИМОЙ ОПТИМИЗАЦИИ

Одним из ключевых аспектов оптимизации производительности графических приложений является эффективное использование ресурсов графического процессора, таких как буферы и массивы вершин. В контексте OpenGL и других графических API процесс работы с буферами может существенно повлиять на производительность визуализаций, особенно в случае динамичных сцен, где необходимо часто обновлять геометрические данные. В традиционных подходах к рендерингу каждый кадр может включать полный пересоздание буферов и пересылку данных на видеокарту, что накладывает значительные затраты на производительность, особенно при высоком числе объектов или сложных сценах. Это связано с высокой стоимостью операций, связанных с выделением памяти и загрузкой данных в видеопамять, что увеличивает общие задержки и нагрузку на систему.

Для повышения производительности разработчики графических приложений ищут способы минимизации количества операций с видеопамятью. Одним из наиболее эффективных подходов является использование предварительной инициализации буферов с последующим их обновлением. Идея заключается в том, чтобы создать буферы и массивы вершин один раз в начале рендеринга и в дальнейшем только изменять их содержимое, что позволяет избежать лишней работы с выделением памяти и загрузкой данных в каждый кадр. В OpenGL это можно реализовать с помощью функции glBufferSubData, которая позволяет обновлять часть данных в уже существующих буферах без необходимости их повторного выделения или полной перезаписи. Это значительно снижает накладные расходы на операции с памятью и ускоряет работу приложения.

Подход, при котором данные обновляются в существующих буферах, а не перезаписываются целиком, является примером машинно-независимой оптимизации, так как его применение не зависит от особенностей конкретной аппаратной платформы или архитектуры. Вместо того чтобы на каждом кадре пересоздавать буферы и заново загружать данные в видеопамять, программа обновляет только измененные части данных, что сводит к минимуму количество операций, требующих взаимодействия с видеокартой. Это особенно эффективно при работе с динамическими объектами, такими как анимации, где геометрия меняется постепенно и частично. Такой подход позволяет существенно снизить время отклика и повысить общую производительность, что делает его важным инструментом в арсенале оптимизаций для графических приложений.

Кроме того, такой способ работы с буферами также может снизить нагрузку на центральный процессор, поскольку уменьшает количество операций с памятью, которые обычно требуют значительных затрат времени. Вместо постоянной переработки данных и их передачи, программа фокусируется на минимизации изменений и обработке только актуальных данных. Это позволяет не только ускорить рендеринг, но и повысить общую отзывчивость системы, что критически важно для приложений с высокой частотой кадров и большими объемами визуальной информации.

# 1.2. ВОЗМОЖНОСТИ ДЛЯ АЛГОРИТМИЧЕСКОЙ ОПТИМИЗАЦИИ

ЗАКОНЧИТЬ

# 2. МАШИННО-НЕЗАВИСИМАЯ ОПТИМИЗАЦИЯ

# 2.1. ОПТИМИЗАЦИЯ БУФЕРОВ

В процессе оптимизации кода программы для решения кубика Рубика была внедрена предварительная инициализация буферов с последующим их обновлением. Этот подход позволил значительно улучшить производительность графического интерфейса, использующего OpenGL и GLFW. Основная идея заключалась в том, чтобы создать и инициализировать буферы один раз при запуске программы, а затем обновлять их содержимое по мере необходимости, вместо создания новых буферов каждый раз при отрисовке элементов.

Для реализации данной оптимизации были созданы функции инициализации буферов, которые выполняют все необходимые операции по созданию и настройке буферов и вершинных массивов. Эти функции вызываются один раз при инициализации окна, что позволяет избежать лишних затрат на создание и настройку буферов в процессе работы программы. В результате, каждый элемент интерфейса, будь то кнопка, квадрат или линия, использует уже существующие буферы, что значительно ускоряет процесс отрисовки.

После инициализации буферов была реализована функция обновления их содержимого. Эта функция использует команду glBufferSubData, которая позволяет обновлять данные в уже существующем буфере, не создавая новый. Таким образом, при изменении состояния элементов интерфейса, например, при нажатии кнопок или изменении цвета квадратов, обновляются только данные в буфере, что значительно снижает нагрузку на систему и улучшает общую производительность.

Оптимизация также включала в себя модификацию функций отрисовки элементов интерфейса, таких как drawButton, drawSquare и drawStrokeLine, для использования предварительно инициализированных буферов. Эти функции были изменены таким образом, чтобы они принимали в качестве параметров указатели на буферы, и обновляли их содержимое по мере необходимости. Это позволило избежать повторной инициализации буферов и сократить количество вызовов OpenGL, что положительно сказалось на производительности.

Также было уменьшено количество вызовов gltInit и gltTerminate из текстовой библиотеки GL Text. Оптимизация заключается в том, чтобы инициализация и завершение библиотеки GLT (GL Text) выполнялись один раз за весь жизненный цикл программы, а не каждый кадр. Это позволяет избежать лишних затрат на повторную инициализацию и освобождение ресурсов, что может значительно улучшить производительность. В результате вызовы gltInit и gltTerminate переносятся из основного цикла отрисовки в начало и конец программы соответственно (для каждого окна по отдельности), что обеспечивает более эффективное использование ресурсов и уменьшает накладные расходы на управление памятью.

# 2.2. ОПТИМИЗАЦИЯ 2

ЗАКОНЧИТЬ

# 3. АЛГОРИТМИЧЕСКАЯ ОПТИМИЗАЦИЯ

ЗАКОНЧИТЬ

# 3.1. ТЕКСТ

ЗАКОНЧИТЬ

# 3.2. ТЕКСТ

ЗАКОНЧИТЬ

# 4. АНАЛИЗ ОПТИМИЗАЦИЙ

В ходе выполнения работы были реализованы две машинно-независимые оптимизации, направленные на улучшение производительности программы, использующей OpenGL для рендеринга кубика Рубика. Для оценки эффективности оптимизаций было проведено сравнение времени отрисовки одного кадра как для главного окна программы, так и для окна ввода кубика. Измерения проводились на выборке из 100 кадров, что обусловлено минимальными колебаниями результатов, при которых разница во времени рендеринга остаётся незначительной. Это позволило получить достаточно точные данные для оценки влияния изменений на производительность.

Первая оптимизация заключалась в инициализации буферов и массивов вершин только один раз в начале рендеринга с последующим обновлением их содержимого через функцию glBufferSubData, а также уменьшении количества вызовов gltInit и gltTerminate из текстовой библиотеки GL Text. Ранее, до применения этой оптимизации, буферы пересоздавались каждый кадр, что приводило к значительным накладным расходам на память и замедляло работу приложения. В результате оптимизации обновление данных происходило гораздо быстрее, так как не требовалось повторно выделять память и передавать данные на видеокарту. Вторая оптимизация касалась ЗАКОНЧИТЬ.

Для оценки изменений в производительности были построены два графика. Первый график (рисунок 1) отображает время отрисовки 100 различных кадров главного окна до и после оптимизации. На графике видно, как улучшения привели к снижению времени рендеринга в среднем на \_\_%. Второй график (рисунок 2) демонстрирует аналогичное сравнение для окна ввода кубика, где также отмечено снижение времени отрисовки кадров после применения оптимизаций в среднем на \_\_%. Эти графики наглядно показывают, как оптимизации снизили время, необходимое для рендеринга, что в свою очередь повышает отзывчивость программы и улучшает пользовательский опыт.

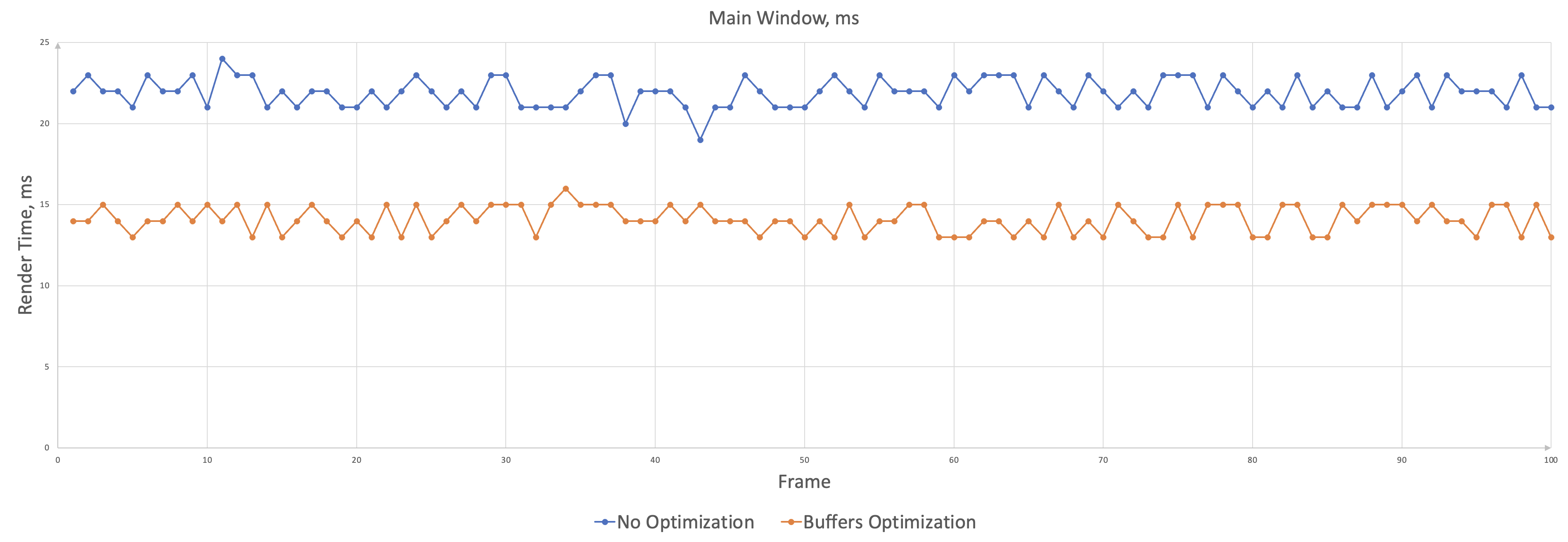


Рисунок 1 – Производительность главного окна

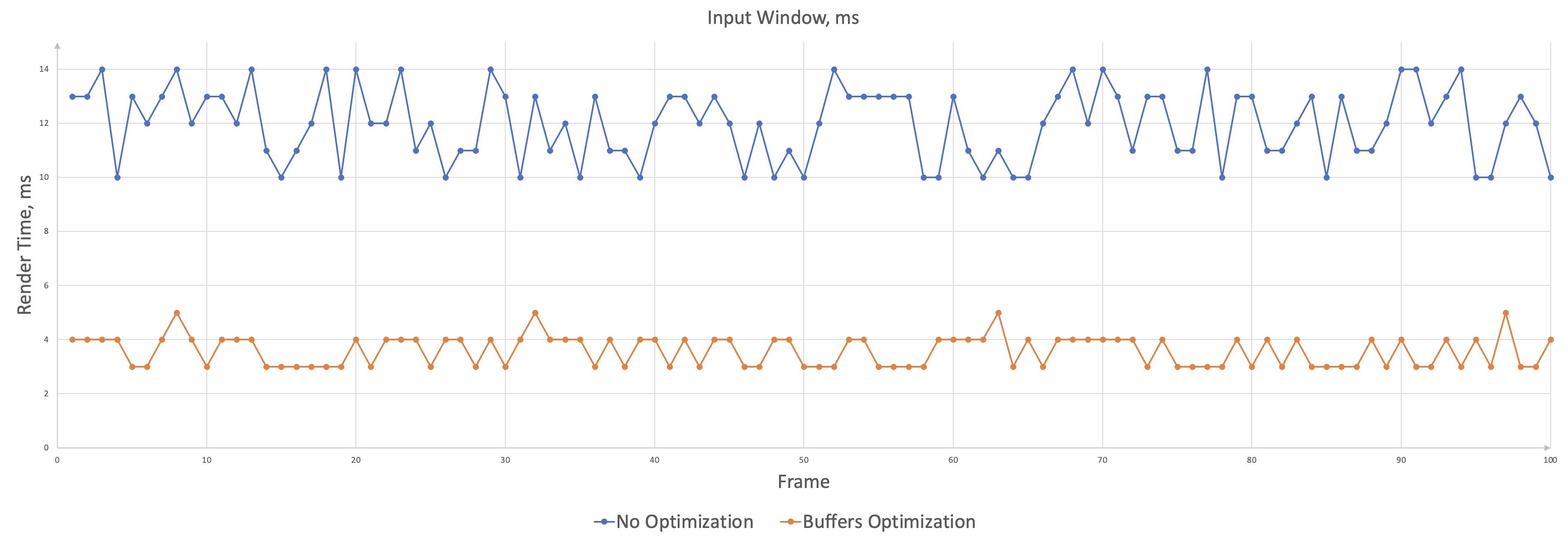


Рисунок 2 – Производительность окна ввода кубика

ЗАКОНЧИТЬ

# ЗАКЛЮЧЕНИЕ

ЗАКОНЧИТЬ

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Buffer Object // OpenGL Wiki. - URL: https://www.khronos.org/opengl/wiki/Buffer\_Object (дата обращения: DD.MM.YYYY).
2. Cube Exproler // Herbert Kociemba. - URL: https://kociemba.org/cube.htm (дата обращения: DD.MM.YYYY).
3. GetOpenFileNameA function (commdlg.h) // Windows App Development. - URL: https://learn.microsoft.com/en-us/windows/win32/api/commdlg/nf-commdlg-getopenfilenamea (дата обращения: DD.MM.YYYY).
4. GetSaveFileNameA function (commdlg.h) // Windows App Development. - URL: https://learn.microsoft.com/en-us/windows/win32/api/commdlg/nf-commdlg-getsavefilenamea (дата обращения: DD.MM.YYYY).
5. Hello Window // LearnOpenGL. - URL: https://learnopengl.com/Getting-started/Hello-Window (дата обращения: DD.MM.YYYY).
6. Input guide // GLFW. - URL: https://www.glfw.org/docs/3.3/input\_guide.html (дата обращения: DD.MM.YYYY).
7. My system for solving Rubik's cube // Jessica Fridrich. - URL: http://www.ws.binghamton.edu/fridrich/system.html (дата обращения: DD.MM.YYYY).
8. OpenGL 4 Reference Pages // Khronos Registry. - URL: https://registry.khronos.org/OpenGL-Refpages/gl4/html/ (дата обращения: DD.MM.YYYY).
9. The Mathematics of the Rubik’s Cube // MIT. - URL: https://web.mit.edu/sp.268/www/rubik.pdf (дата обращения: DD.MM.YYYY).
10. The OpenGL Graphics System: A Specification // The Khronos Group Inc. - URL: https://registry.khronos.org/OpenGL/specs/gl/glspec40.core.pdf (дата обращения: DD.MM.YYYY).
11. WCA Regulations // World Cube Association. - URL: https://www.worldcubeassociation.org/regulations/ (дата обращения: DD.MM.YYYY).
12. Window guide // GLFW. - URL: https://www.glfw.org/docs/3.3/window\_guide.html (дата обращения: DD.MM.YYYY).
13. Zenity Manual // The GNOME Project. - URL: https://help.gnome.org/users/zenity/stable/ (дата обращения: DD.MM.YYYY).

# ПРИЛОЖЕНИЕ А. БЛОК-СХЕМЫ ФУНКЦИЙ ОТРИСОВКИ ОКОН

ЗАКОНЧИТЬ

# ПРИЛОЖЕНИЕ Б. ЛИСТИНГИ ФУНКЦИЙ ОТРИСОВКИ ОКОН

ТЕКСТ

ЗАКОНЧИТЬ

ТЕКСТ

ЗАКОНЧИТЬ

# ПРИЛОЖЕНИЕ В. БЛОК-СХЕМЫ КЛЮЧЕВЫХ ФУНКЦИЙ АЛГОРИТМА РЕШЕНИЯ КУБИКА

ЗАКОНЧИТЬ

# ПРИЛОЖЕНИЕ Г. ЛИСТИНГИ КЛЮЧЕВЫХ ФУНКЦИЙ АЛГОРИТМА РЕШЕНИЯ КУБИКА

ТЕКСТ

ЗАКОНЧИТЬ

ТЕКСТ

ЗАКОНЧИТЬ