Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 20Б.11-мм

Пушкин Тимофей Дмитриевич

Реализация кросс-платформенной библиотеки нахождения глубины изображения по стереопаре

Отчёт по учебной практике
в форме «Производственное задание»

Научный руководитель:
старший преподаватель каф. СП Я.А. Кириленко

Консультанты:
инженер-исследователь Сколтех А.В. Корнилова
Android-разработчик VK Д.С. Ярош

Санкт-Петербург
2022

Оглавление

1 Введение

Решения таких задач как создание 3D-реконструкции сцены, оцифровка объектов реального мира, расположение виртуальных предметов в дополненной реальности и коррекция фотоснимков с учетом глубины (эффект боке) имеют большой спрос в современном мире. Названные задачи входят в широкий спектр проблем, решаемых с помощью вычисления карты глубины сцены — информации об удаленности объекта на снимке от точки, с которой данный снимок был снят.

Для нахождения карт глубины активно используются такие технологии как, например, LiDAR [13] или Time-of-Flight [14], однако для их использования требуется специальное оборудование, не имеющее широкого распространения и зачастую требующее дополнительных навыков для использования. Более доступным вариантом являются подходы для вычисления глубины с использованием снимков с обыкновенных камер, к примеру, камер мобильных устройств. В наше время смарфтоны широко распространены и способны проводить достаточно качественную съемку, что делает их удобным инструментом для использования в задачах, связанных с захватом изображения. Как показано в [10], современные смартфоны обладают достаточной вычислительной мощностью, чтобы выполнять операции, необходимые для нахождения карт глубины в мягком реальном времени (soft real-time).

Существуют различные методы нахождения карты глубины на смартфонах. Студент четвертого курса направления «Программная инженерия» СПбГУ Азат Ахметьянов в своей выпускной квалификационной работе исследовал данные методы и предложил свой подход для получения глубины изображения из стереопары[1] для мобильных устройств, основанный на алгоритме Stereo Semi-Global Block Matching (StereoSGBM) [15] из библиотеки OpenCV [5]. Использование StereoSGBM привело к наилучшим результатам в сравнении с остальными рассмотренными методами. Реализация данного подхода, полу-

[1]Стереопара — совокупность двух изображений одного и того же объекта, полученных с различных точек съемки

ченная Азатом Ахметьяновым в ходе его исследования, была создана с целью демонстрации качественного превосходства алгоритма. Она не ориентирована на быстродействие и подсчет глубины в мягком реальном времени, а также не предназначена для интеграции в сторонние приложения — в частности, для использования на мобильных устройствах.

В данной работе описывается создание кросс-платформенной библиотеки, основанной на предложенной Азатом Ахметьяновым реализации его подхода к подсчету карты глубины из стереопары с использованием алгоритма StereoSGBM из OpenCV. Создаваемая библиотека рассчитана на запуск вычислений на мобильных устройствах и подсчет глубины изображения в мягком реальном времени. Таким образом, она не будет обладать описанными ранее недостатками оригинальной реализации выбранного подхода.

2 Цель и задачи

Целью данной работы является создание кросс-платформенной библиотеки для нахождения карты глубины по стереопаре с применением алгоритма StereoSGBM из библиотеки OpenCV. Для выполнения названной цели были поставлены следующие задачи:

1. Выполнить обзор механизмов использования OpenCV на различных платформах.

2. Реализовать кросс-платформенную библиотеку на основе демонстрационной версии подхода.

3. Провести качественное сравнение полученной реализации с оригинальной.

4. Провести апробацию полученной библиотеки, реализовав на её основе мобильное приложение для нахождения карты глубины по стереопаре.

5. Провести проверку возможности применения полученной реализации для вычисления карт глубины в мягком реальном времени на мобильных устройствах.

3 Обзор

В данном разделе описывается подход к вычислению карты глубины, взятый за основу реализуемой библиотеки, а также рассматриваются механизмы использования библиотеки OpenCV и связанные с ними сторонние инструменты.

3.1 Описание использованного подхода и оригинальной реализации

Входными данными для нахождения карты глубины являются два изображения (стереопара) и калибровочные параметры[2], необходимые для предварительной коррекции изображений и вычисления глубины.

Оригинальная реализация подхода была написана на Python. Она основана на библиотеке OpenCV, в частности, на входящем в её состав алгоритме StereoSGBM, и осуществляет построение карты глубины следующим образом:

1. Полученные на вход изображения выравниваются относительно друг друга с использованием калибровочных параметров.

2. Выровненные изображения приводятся к заранее указанному размеру[3] (масштабируются).

3. Вычисляется взаимное смещение точек на изображениях — для этого используется StereoSGBM.

4. По найденному смещению в пространстве строится карта глубины — для этого применяются данные о взаимном расположении камер, запечатлевших стереопару.

Входные данные в оригинальной реализации хранятся в форматах, поддерживаемых в OpenCV.

[2]Калибровочные параметры состоят из набора матриц, используемых для выравнивания изображений, а также содержащих информацию о взаимном расположении камер

[3]Предполагается, что указанный размер будет меньше оригинального — тогда карта глубины будет вычисляться быстрее

3.2 Механизмы использования OpenCV

Библиотека OpenCV имеет несколько видов поставки, среди которых есть отдельные собранные со стандартными параметрами комплекты разработки (SDK) для iOS и Android. Альтернативно имеется возможность собирать её самостоятельно из исходного кода, указывая необходимые параметры сборки.

3.2.1 SDK для мобильных платформ

Собранные версии OpenCV для мобильных платформ помимо самой библиотеки, написанной на C++, включают в себя интерфейсы для удобного взаимодействия с ней из языков, используемых на соответствующих платформах: Java/Kotlin для Android и Objective-C/Swift для iOS. Данные интерфейсы представляют из себя набор классов-оберток: вызов их методов переводится в вызов соответствующего C++-кода. Из-за этого, согласно [9], на Android частые обращения к Java-интерфейсу могут повлечь проблемы с производительностью.

Также стоит отметить, что использование в библиотеке интерфейсов из данных SDK не позволяет получить кросс-платформенную реализацию.

3.2.2 Сборка из исходного кода

Сборка OpenCV из исходного когда также является одним из рекомендуемых способов работы с C++-реализацией библиотеки.

При самостоятельной сборке появляется возможность пользоваться широким спектром конфигурационных параметров библиотеки. С помощью них можно исключать из сборки части OpenCV, не используемые в конкретном проекте, а также наоборот включать оптимизации, отключённые по умолчанию.

К негативным сторонам самостоятельной сборки стоит отнести сложность её автоматизации при встраивании в процесс сборки проекта: хотя OpenCV и использует CMake [6], её подключение при помощи автоматической загрузки и сборки через CMake официально не под-

держивается. Таким образом, рекомендуемым способом подключения OpenCV с особой конфигурацией является её предварительная отдельная сборка и последующее подключение к проекту.

3.3 Генерация платформозависимых интерфейсов

В случае отказа от использования готовых интерфейсов OpenCV и реализации библиотеки на C++ возникает необходимость самостоятельного создания её интерфейсов для Java/Kotlin на Android и Objective-C/Swift на iOS для упрощения работы с ней на данных платформах.

Для вызова C/C++ кода в Java/Kotlin применяется JNI [8], а в Objective-C/Swift — прослойка на Objective-C++. В случае самостоятельной разработки интерфейсов потребуется реализовывать их для каждой платформы отдельно, а также тратить ресурсы на их последующую поддержку. Избавиться от обозначенных проблем позволяют инструменты для автоматической генерации интерфейсов для C/C++ кода, которые будут рассмотрены далее.

Djinni [7] и **SWIG** [11] — данные инструменты позволяют генерировать интерфейсы для Java и Objective-C (а также некоторых других языков) по созданному вручную файлу-интерфейсу с описанием C++ кода в специальном формате, специфичном для конкретного инструмента. SWIG также имеет расширенную поддержку в CMake. Разработка и сопровождение Djinni на данный момент прекращены.

Scapix [12] — в отличие от Djinni и SWIG, данный инструмент позволяет генерировать интерфейсы для Java и Objective-C в частности без написания каких-либо файлов-интерфейсов. Однако, для его использования требуется некоторое изменение исходного кода проекта: необходимо объявить классы, для которых надлежит сгенерировать интерфейс, наследниками служебного класса Scapix.

4 Разработка библиотеки

При разработке библиотеки было принято решение отказаться от использования платформозависимых интерфейсов, предоставляемых библиотекой OpenCV. Во-первых, это сопосбствует кроссплатформенности, а во-вторых, позволяет избежать ухудшения производительности на Java-платформах[4]. Таким образом, было решено разрабатывать библиотеку на C++ как на языке, позволяющем обеспечить поддержку наиболее обширного множества платформ и взаимодействать с OpenCV без дополнительных интерфейсов.

Для генерации платформозависимых интерфейсов применялся Scapix, так как он требует наименьшей конфигурации, а необходимое при его использовании изменение исходного кода библиотеки не является критичным.

4.1 Архитектура библиотеки

Функциональность библиотеки — подсчет карты глубины по стереопаре и калибровочным параметрам — была реализована посредством единого класса `DepthEstimator`.

В своем конструкторе `DepthEstimator` принимает путь к файлу с калибровочными параметрами. Было решено передавать именно путь к файлу, а не сами параметры, из-за их большого объема[5], который бы иначе передавался через платформозависимый интерфейс. В будущем планируется расширить библиотеку дополнительным классом, отвечающим за автоматическую калибровку двух камер, то есть за вычисление калибровочных параметров, используемых в `DepthEstimator`.

Вычисление глубины регулируется набором параметров, выставляемых через отдельные методы. По умолчанию для всех них устанавливаются значения, выбранные в оригинальной реализации. Доступными

[4]При создании интерфейса для библиотеки в целом многократные вызовы через Java-интерфейс для каждого обращения к OpenCV будут заменены всего двумя вызовами: обращением к библиотеке и возвращением результата из неё

[5]Для стереопары с разрешением 1920 на 1080 пикселей калибровочные параметры занимают более 50 Мегабайт

для регулировки сделаны параметры, не предопределенные используемым подходом. В ходе реализации библиотеки была преобразована логика изменения размера изображений: вместо задания конкретного размера итоговый размер определяется долей от оригинального с сохранением пропорций, то есть задание происходит посредством выставления коэфициента масштабирования (по умолчанию данный параметр равен единице, то есть масштабирование не проводится). Помимо этого была добавлена автоматическая регулировка параметров, зависящих от масштабирования: пользователю достаточно выставить все параметры для исходного размера изображения, а в случае масштабирования они будут автоматически отрегулированы соответствующим образом.

Построение карты глубины осуществляется посредством метода `estimateDepth` — он вычисляет карту глубины по двум изображениям и возвращает её в виде двумерного массива чисел с плавающей запятой. Так как от реализации требовалась поддержка возможности подсчета карты глубины в мягком реальном времени, объем вычислений в данном методе был минимизирован: в отличие от оригинальной реализации, объект класса `StereoSGBM`, используемый для нахождения смещения точек стереопары, создается лишь однажды при конструировании объекта `DepthEstimator`, а затем многократно используется в `estimateDepth`, что избавляет от задержек на его создание.

4.2 Сборка библиотеки

Разработанная библиотека является CMake-проектом.

Библиотеку OpenCV было принято решение собирать для проекта самостоятельно из-за возможности более гибкой её настройки в таком случае. Однако, так как интеграция сборки OpenCV из исходного кода в процесс сборки других CMake-проектов официально не поддерживается, для этого был создан отдельный CMake-скрипт. Данный скрипт проводит загрузку и сборку OpenCV, подключая при этом лишь используемые в библиотеке модули и оптимизации. При его использовании можно как применить C/C++ компиляторы, выбранные в системе

по умолчанию, что удобно при сборке для ПК, так и указать конкретный набор инструментов (toolchain) для применения, что требуется при сборке для мобильных платформ.

В качестве поддержана возможность использовать OpenCV, установленный в системе, однако в таком случае в собранном решении могут отсутствовать некоторые оптимизации из-за их отключения в используемой сборке OpenCV.

Таким образом, разработанная библиотека собирается с использованием ранее скомпонованной OpenCV, поиск которой в системе осуществляется средствами CMake. При сборке также есть возможность включить генерацию интерфейса библиотеки для языков, поддерживаемых Scapix.

5 Качественная оценка

Разработанная реализация была проверена на корректность посредством сравнения получаемых с её помощью результатов с результатами от оригинальной реализации.

5.1 Условия тестирования

Для проверки использовался набор из четырех стереопар, снятых в различных сценах. Две из них имеют калибровочные параметры с расстоянием между камерами, равным 10 сантиметрам, а две другие — 14 сантиметрам. Изображения в стереопарах имеют разрешение 1920 на 1080 пикселей. Данный набор был получен и предоставлен Азатом Ахметьяновым.

Сборка и тестирование проводились на ПК под управлением Ubuntu 20.04.4 64-bit (Linux 5.13.0-41-generic x86_64). Реализованная библиотека и использованная в ней OpenCV собирались при помощи gcc 9.4.0. Оригинальная реализация запускалась на Python 3.8.10.

5.2 Метод тестирования

Результаты, выдаваемые реализованной библиотекой, проверялись на соответствие картам глубины, получаемым при помощи оригинальной реализацией подхода, через подсчёт абсолютных и относительных разностей значений в каждой точке сравниваемых карт глубины. Сопоставляемые карты глубины вычислялись без ограничений по значениям, масштабирование не применялось, остальные параметры имели значения по умолчанию (одинаковые для обеих реализаций).

5.3 Результаты

В Таблице 1 приведены полученные результаты. Также на Рис. 1 продемонстрированы нормализованные карты глубины одной из стереопар для визуального сравнения.

Стереопара	Абсолютная разность	Относительная разность
1	0.04 ± 0.08 мм	$1.4 \cdot 10^{-6} \pm 8 \cdot 10^{-7}$
2	0.09 ± 0.10 мм	$1.3 \cdot 10^{-6} \pm 8 \cdot 10^{-7}$
3	0.04 ± 0.06 мм	$0.8 \cdot 10^{-6} \pm 6 \cdot 10^{-7}$
4	0.06 ± 0.07 мм	$0.6 \cdot 10^{-6} \pm 5 \cdot 10^{-7}$

Таблица 1: Разности значений в соответствующих точках карт глубины, подсчитанных при помощи полученной и оригинальной реализаций; значения указаны в форме среднее значение $\pm$ стандартное отклонение.

Рис. 1: Слева направо: изображение из стереопары (левое), карта глубины от оригинальной реализации, карта глубины от полученной реализаций.

Полученные несоответствия являются следствием обработки карты взаимного смещения точек стереопары перед вычислением по ней карты глубины: из-за особенностей реализации StereoSGBM (с помощью которого вычисляется карта взаимного смещения точек стереопары) все значения карты необходимо перед подсчетом глубины делить на 16 — в ходе данной корректировки и появляются выявленные несоответствия. Это объясняется различиями в работе с числами с плавающей запятой (из которых и состоят карты глубины) в Python и C++: в Python применяются 64-битные значения, в то время как в методах C++-реализации OpenCV, используемых при расчетах глубины, — 32-битные.

Итого, во-первых, найденные в ходе проверки абсолютные значения несоответствий измеряются долями миллиметра, поэтому они не должны влиять на дальнейшее применение карт глубины в прикладных задачах. А во-вторых, как показано в работе Азата Ахметьянова, средняя относительная погрешность рассматриваемого метода подсчета карты глубины примерно равна 0.1 — найденные в ходе проверки значения относительной разности пренебрежимо малы по сравнению с

данной погрешностью. Таким образом, справедливо считать, что разработанная реализация соответствует оригинальной.

6 Апробация на Android

Для проведения апробации полученной C++-библиотеки на её основе были созданы Android-библиотека, позволяющая удобно применять разработанный инструмент в Android, и Android-приложение, с помощью которого можно визуально оценить результаты работы полученной реализации, а также замерить скорость её работы.

6.1 Детали реализации

Библиотека и приложение поддерживают Android API 21 и выше — это версии, поддерживаемые в библиотеке OpenCV для всех Android ABI [2].

Android-библиотека автоматически собирается из C++-кода для всех ABI, используемых в Android, с помощью Android Gradle Plugin [3]. Она представляет из себя Java-интерфейс C++-реализации, автоматически генерируемый при сборке с помощью Scapix.

Android-приложение разработано на языке Kotlin. Оно позволяет выбрать на устройстве файл с конфигурационными параметрами и стереопару, по которым при помощи реализованной библиотеки подсчитывается карта глубины. Также есть возможность указать масштаб, к которому будут приведены изображения стереопары перед подсчетами, — остальные параметры, определенные в реализованной библиотеке, оставлены со значениями по умолчанию. После подсчета карты глубины на экран устройства выводится черно-белое изображение, демонстрирующее найденную глубину для каждого пикселя по градации серого цвета, — визуализация полученной карты глубины, — а также время её подсчета. Интерфейс приложения продемонстрирован на Рис. 2.

6.2 Оценка производительности

Для проверки возможности применения полученного решения для вычисления карт глубины в мягком реальном времени на мобильных

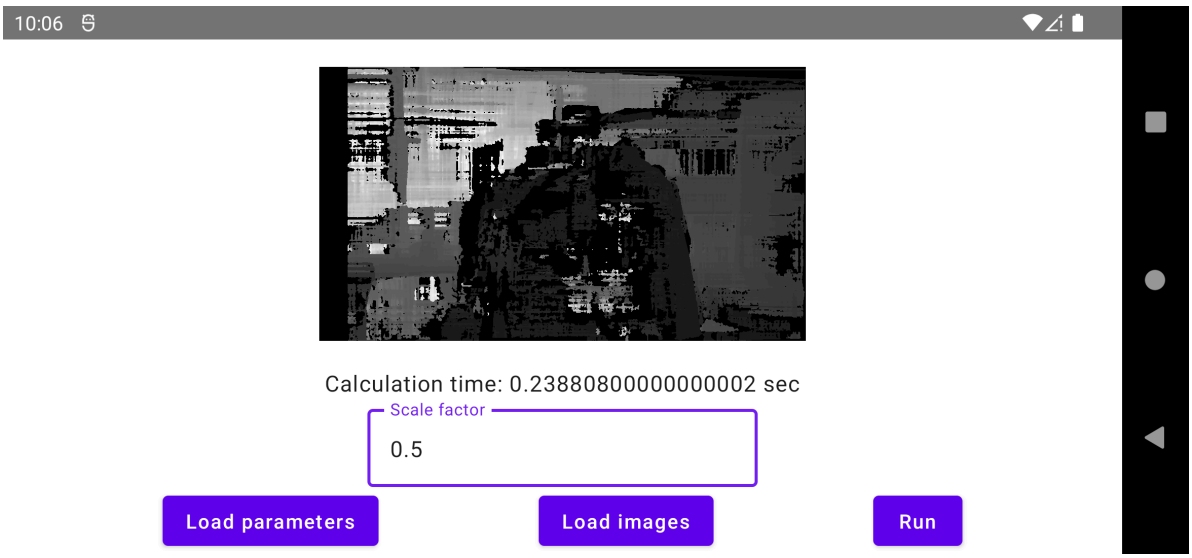

Рис. 2: Графический интерфейс реализованного Android-приложения после подсчета карты глубины.

устройствах был проведен подсчет времени, необходимого для получения карты глубины на Android-смартфонах с различным предварительным масштабированием.

6.2.1 Условия проведения оценки

При оценке применялся набор стереопар, описанный в секции о качественной оценке реализации.

Сборка Android-приложения и библиотеки осуществлялась на ПК под управлением Windows 10 19044.1706 64-bit: использовалась release-версия со стандартными оптимизациями, применяемыми для неё в Android Gradle Plugin, а также отключенным логгированием. C++-библиотека и используемая в ней OpenCV собирались при помощи инструментов, предоставленных в Android Native Development Kit [4] версии r23b.

Замеры проводились на следующих смартфонах:

- Xiaomi Mi 5 (Android API level 30, LineageOS 18.1, версия прошивки — lineage-18.1-20220515-nightly-gemini),
- Samsung Galaxy S10e SM-G970F/DS (Android API level 31, One UI 4.0, версия прошивки — G970FXXUEGULB),

- Samsung Galaxy S10 Lite SM-G770F/DS (Android API level 31, One UI 4.0, версия прошивки — G770FXXU6FUL5).

Все указанные устройства имеют Android ABI arm64-v8a с поддержкой Neon [1], соответственно, библиотека и OpenCV собирались для данного ABI со включёнными Neon-оптимизациями.

6.2.2 Метод проведения оценки

Производительность оценивалась через измерение времени вычисления карты глубины для стереопары в реализованном приложении. Измерялось время выполнения следующих действий:

1. Вызов приложением C++-реализации библиотеки через JNI.
2. Вычисление библиотекой карты глубины.
3. Получение результата из библиотеки приложением.

Карты глубины вычислялись на различных разрешениях, полученных при помощи предварительного масштабирования: 240p, 360p, 480p, 720p и 1080p (без масштабирования) — целью данных замеров является демонстрация влияния предварительного масштабирования на скорость работы реализации. На каждом из использованных смартфонов проводились замеры для каждой комбинации разрешения и стереопары.

6.2.3 Результаты

На Рис. 3 приведен график полученных результатов. Также на Рис. 4 продемонстрированы нормализованные карты глубины одной из стереопар в использованных при оценке разрешениях для визуального сравнения.

Как видно из результатов оценки производительности, полученную реализацию можно применять на современных Android-смартфонах для вычисления карт глубины в «мягком» реальном времени, а для

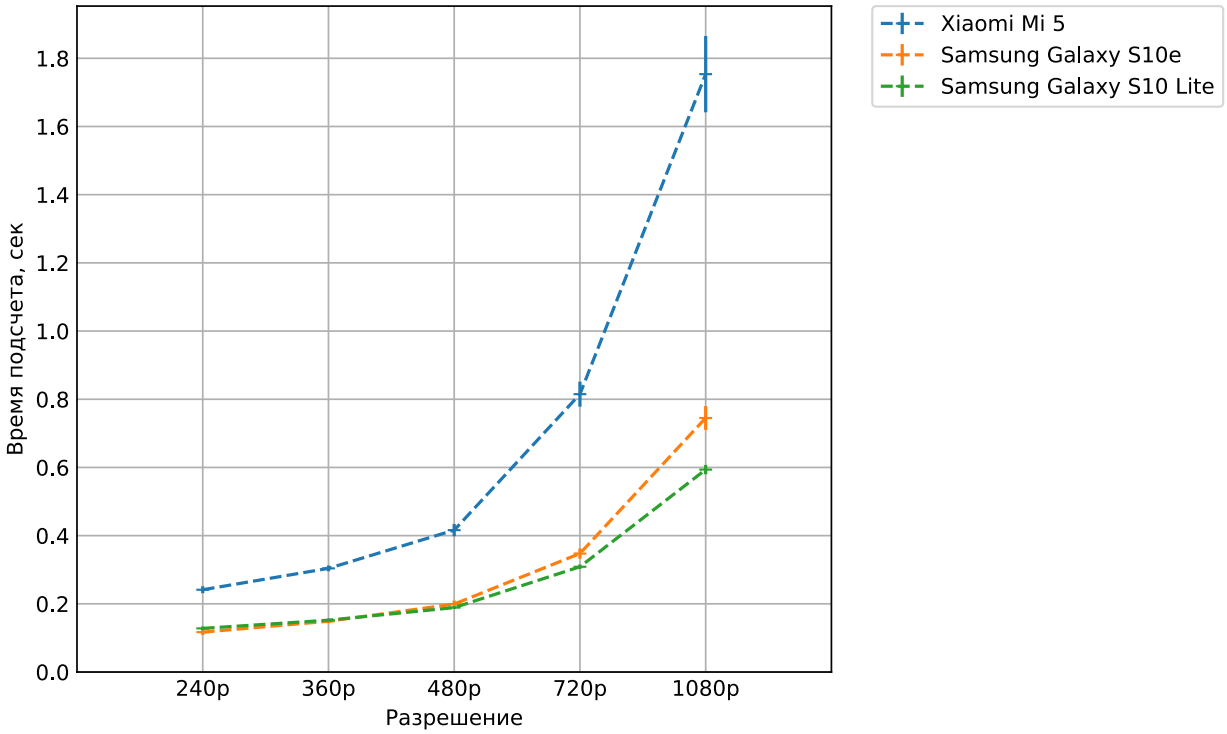

Рис. 3: График зависимости времени вычисления карты глубины от разрешения стереопары; на график нанесены средние значения со среднеквадратичным отклонением, подсчитанные за 50 замеров каждое; меньше — лучше.

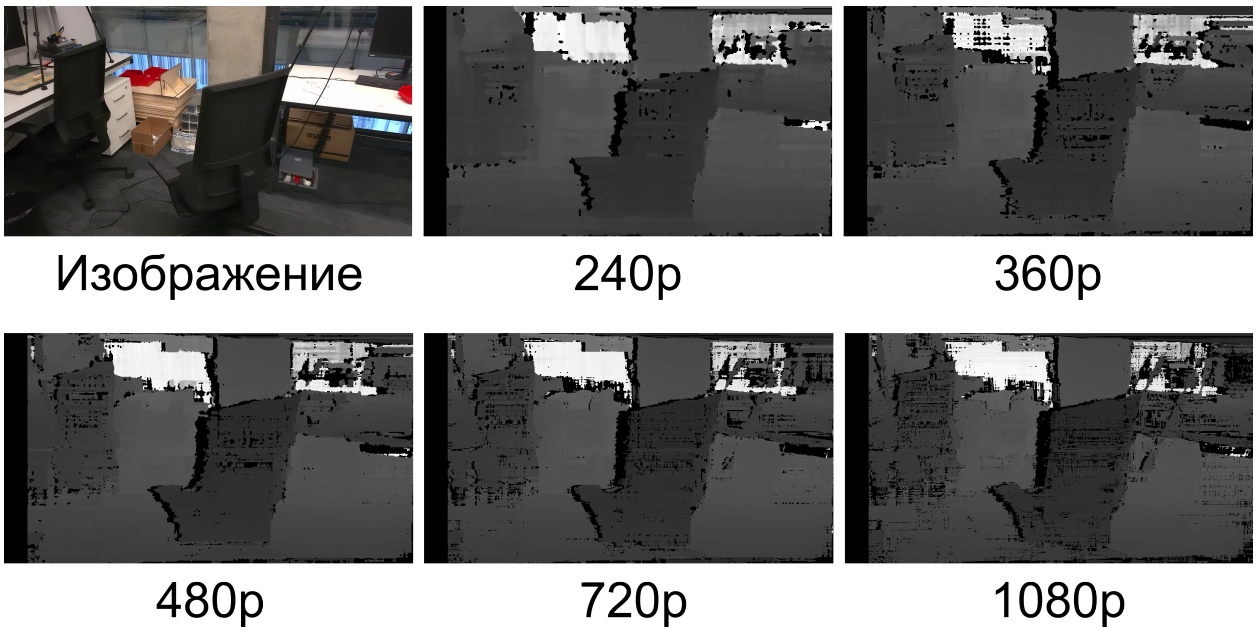

Рис. 4: Карты глубины, вычисленные при различном предварительном масштабировании; для лучшей визуализации все изображения приведены к разрешению 1080p средствами стороннего программного обеспечения с интерполяцией через ближайшее известное значение.

повышения скорости вычислений есть возможность предварительно уменьшать разрешение средствами, предоставляемыми реализованной библиотекой. Однако, продемонстрированной скорости вычислений может быть недостаточно, например, для создания видео-записей с комфортной для человека кадровой частотой.

7 Заключение

Были получены следующие результаты.

1. Выполнен обзор механизмов использования OpenCV на различных платформах.
2. Реализована кросс-платформенная библиотека на основе демонстрационной версии подхода к нахождению карты глубины по стереопаре.
3. Проведено качественное сравнение полученной реализации с оригинальной.
4. Проведена апробация полученной библиотеки через реализацию на её основе мобильного приложения для нахождения карты глубины по стереопаре.
5. Проведена проверка возможности применения полученной реализации для вычисления карт глубины в мягком реальном времени на мобильных устройствах.

Результаты работы расположены в данном репозитории.

7.1 Направления будущей работы

Реализованная в данной работе библиотека требует заранее подготовленных сторонними средствами калибровочных параметров. Планируется расширить её функциональность, добавив возможность автоматического нахождения данных параметров.

7.2 Благодарность

Выражается благодарность студенту четвертого курса направления «Программной инженерия» СПбГУ Азату Ришатовичу Ахметьянову[6],

[6] https://github.com/azaat

автору подхода к нахождению карты глубины по стереопаре, на котором основана данная работа, за обширные консультации по устройству подхода и его демонстрационной реализации, а также за предоставление набора данных для проведения тестирования полученной библиотеки.

Также выражается благодарность студенту второго курса направления «Технологии программирования» СПбГУ Павлу Евгеньевичу Мокееву[7] за подробный обзор инструментов для автоматической генерации интерфейсов C/C++-кода.

[7] https://github.com/pmokeev

Список литературы

[1] ARM NEON. — URL: https://www.arm.com/technologies/neon (online; accessed: 15.05.2022).

[2] Android ABIs. — URL: https://developer.android.com/ndk/guides/abis (online; accessed: 16.05.2022).

[3] Android Gradle plugin release notes. — URL: https://developer.android.com/studio/releases/gradle-plugin (online; accessed: 11.05.2022).

[4] Android NDK. — URL: https://developer.android.com/ndk (online; accessed: 18.05.2022).

[5] Bradski G. The OpenCV Library // Dr. Dobb's Journal of Software Tools. — 2000.

[6] CMake. — URL: https://cmake.org (online; accessed: 11.05.2022).

[7] Djinni. — URL: https://github.com/dropbox/djinni (online; accessed: 11.05.2022).

[8] JNI. — URL: https://docs.oracle.com/javase/8/docs/technotes/guides/jni (online; accessed: 11.05.2022).

[9] OpenCV Android Best Practices. — URL: https://opencv.org/android-best-practices (online; accessed: 09.05.2022).

[10] Real-Time Single Image Depth Perception in the Wild with Handheld Devices / Filippo Aleotti, Giulio Zaccaroni, Luca Bartolomei et al. // Sensors. — 2021. — Vol. 21, no. 1. — URL: https://www.mdpi.com/1424-8220/21/1/15.

[11] SWIG. — URL: https://www.swig.org (online; accessed: 11.05.2022).

[12] Scapix. — URL: https://github.com/scapix-com/scapix (online; accessed: 11.05.2022).

[13] Wikipedia contributors. Lidar — Wikipedia, The Free Encyclopedia. — 2004. — URL: https://en.wikipedia.org/wiki/Lidar (online; accessed: 11.05.2022).

[14] Wikipedia contributors. Time of flight — Wikipedia, The Free Encyclopedia. — 2004. — URL: https://en.wikipedia.org/wiki/Time_of_flight (online; accessed: 11.05.2022).

[15] cv::StereoSGBM Class Reference. — URL: https://docs.opencv.org/4.5.5/d2/d85/classcv_1_1StereoSGBM.html (online; accessed: 11.05.2022).