МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»

Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой ЭВМ
_____ Д.В.Топольский
«___»_____ 2023 г.

МОБИЛЬНОЕ ПРИЛОЖЕНИЕ ДЛЯ РАСЧЁТА МАТЕРИАЛОВ ЧИСТОВОЙ ОТДЕЛКИ ЖИЛЫХ ПОМЕЩЕНИЙ

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУРГУ-090301.2023.039 ПЗ ВКП

Руководитель работы,
к. пед. н., доцент каф. ЭВМ
_____М.А.Алтухова
«___»_____ 2023 г.

Автор работы,
студент группы КЭ-405
_____Л.Р.Тодуа
«___»_____ 2023 г.

Нормоконтролёр,
ст. препод. каф. ЭВМ
_____С.В.Сяськов
«___»_____ 2023 г.

Челябинск-2023

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»

Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

УТВЕРЖДАЮ
Заведующий кафедрой ЭВМ
_____ Д.В.Топольский
«___»_____ 2023 г.

ЗАДАНИЕ

на выпускную квалификационную работу бакалавра

студенту группы КЭ-405
Тодуа Левани Романовичу,
обучающемуся по направлению
09.03.01 «Информатика и вычислительная техника»

1. **Тема работы: «**Мобильное приложение для расчёта материалов чистовой отделки жилых помещений» утверждена приказом по университету от «25» апреля 2023 г. № 753-13/12

2. **Срок сдачи студентом законченной работы:** «___» _____ 2023 г.

3. **Исходные данные к работе:**

3.1. Разработать мобильное приложение для широкого круга пользователей, предназначенное для облегчения задачи выбора и расчёта количества материалов для внутренней отделки помещения с возможностью визуализации выбранного варианта на трёхмерной модели помещения.

3.2. Обеспечить основной функционал приложения:

– построение трёхмерной модели помещения;

– получение текстуры обоев с помощью фото;

– заполнение модели текстурами в соответствии с размерами и узором обоев;

– возможность сравнения вариантов;

– расчёт количества материалов.

3.3. Нефункциональные требования:

– доступность широкому кругу пользователей без оплаты и авторизации;

– независимость приложения от конкретных поставщиков или производителей отделочных материалов.

4. **Перечень подлежащих разработке вопросов:**

В рамках выполнения выпускной квалификационной работы необходимо проработать следующие вопросы:

1) аналитический обзор аналогов по тематике работы;

2) разработка алгоритмов обработки изображения обоев и построения трёхмерных визуализаций жилых помещений с наложением текстур;

3) проектирование интерфейса приложения;

4) разработка и отладка приложения;

5) тестирование приложения.

5. **Дата выдачи задания:** «___» _____ 2023 г.

Руководитель работы_____/ М.А. Алтухова /

Студент _____/ Л.Р. Тодуа /

КАЛЕНДАРНЫЙ ПЛАН

Этап	**Срок сдачи**	**Подпись руководителя**
Аналитический обзор аналогов по тематике работы	01.03.2023	
Разработка алгоритмов обработки изображения обоев и построения трёхмерных визуализаций жилых помещений с наложением текстур	10.04.2023	
Проектирование интерфейса приложения	15.04.2023	
Разработка и отладка приложения	25.04.2023	
Тестирование приложения	10.05.2023	
Компоновка текста работы и сдача на нормоконтроль	16.05.2023	
Подготовка презентации и доклада	20.05.2023	

Руководитель работы _____/ М.А. Алтухова /

Студент _____/ Л.Р. Тодуа /

АННОТАЦИЯ

Л.Р. Тодуа. Мобильное приложение для расчёта материалов чистовой отделки жилых помещений. – Челябинск: ФГАОУ ВО «ЮУрГУ (НИУ)», ВШ ЭКН; 2023, 59 с., 15 ил., библиогр. список – 12 наим.

Выпускная квалификационная работа посвящена разработке мобильного приложения для расчёта материалов чистовой отделки жилых помещений. Цель работы заключается в создании удобного инструмента, который поможет пользователям выбирать и рассчитывать необходимое количество отделочных материалов, а также визуализировать выбранный вариант на трёхмерной модели помещения.

В рамках работы были рассмотрены аналоги по данной тематике, разработаны алгоритмы обработки фотографических изображений обоев и построения 3D-визуализаций жилых помещений с наложением текстур. Также спроектирован интерфейс приложения, произведена разработка и отладка, проведено тестирование.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ 7
1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 9
1.1 Обзор и сравнение аналогов 9
1.2 Определение требований 12
1.3 Анализ основных технологических решений 13
2 РАЗРАБОТКА АЛГОРИТМОВ ОБРАБОТКИ ИЗОБРАЖЕНИЯ ОБОЕВ И ПОСТРОЕНИЯ ТРЁХМЕРНЫХ ВИЗУАЛИЗАЦИЙ ЖИЛЫХ ПОМЕЩЕНИЙ С НАЛОЖЕНИЕМ ТЕКСТУР 17
2.1 Разработка алгоритма обработки изображения обоев 17
2.2 Разработка алгоритма построения трёхмерных визуализаций жилых помещений с наложением текстур 20
3 ПРОЕКТИРОВАНИЕ ИНТЕРФЕЙСА ПРИЛОЖЕНИЯ 25
4 РАЗРАБОТКА И ОТЛАДКА ПРИЛОЖЕНИЯ 27
4.1 Разработка приложения 27
4.2 Отладка приложения 37
5 ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ 39
5.1 Планирование тестирования приложения 39
5.2 Проведение тестирования приложения 40
ЗАКЛЮЧЕНИЕ 44
БИБЛИОГРАФИЧЕСКИЙ СПИСОК 46
ПРИЛОЖЕНИЯ 48
ПРИЛОЖЕНИЕ А. Исходный код класса активности редактора трёхмерной модели помещения 48
ПРИЛОЖЕНИЕ Б. Исходный код класса средства визуализации 53
ПРИЛОЖЕНИЕ В. Исходный код класса активности создания трёхмерной модели помещения 58

ВВЕДЕНИЕ

Чистовая отделка жилых помещений является заключительным этапом ремонтных работ. При чистовой отделке на пол укладывают различные покрытия: паркет, ламинат, керамическую или керамогранитную плитку. А отделка стен и потолка охватывает такие виды деятельности, как окраска, нанесение декоративной штукатурки, облицовка кафелем и покрытие обоями [1].

Актуальность данной темы заключается в том, что в настоящее время имеется категория людей, которые занимаются ремонтом самостоятельно, не привлекая к работе специалистов данной области. В частности, это касается чистовой отделки помещения, так как она может быть менее сложной в реализации для обычного человека, чем черновая отделка. На данном этапе могут возникнуть проблемы, которые обычному человеку трудно решить без привлечения и оплаты специалистов. Человеку необходимо понимать то, как может выглядеть готовый вариант помещения с разными материалами и какое количество материала может быть необходимо для работы.

Для решения этой проблемы были разработаны специальные мобильные приложения, способные реализовать расчёт необходимого количества материалов, построить трёхмерную визуализацию помещения с наложением определённых текстур. Однако на данный момент не было обнаружено мобильного приложения, способного объединить описанный функционал с возможностью накладывать текстуры, не зависящие от конкретных поставщиков или производителей отделочных материалов.

Целью представленной выпускной квалификационной работы является разработка мобильного приложения для широкого круга пользователей, которое будет предназначено для облегчения задачи выбора и расчёта количества материалов для внутренней отделки помещения с возможностью визуализации выбранного варианта на трёхмерной модели помещения.

Для достижения поставленной цели, необходимо решить следующие поставленные задачи:

1) провести аналитический обзор аналогов по тематике работы;

2) разработать алгоритмы обработки фотографического изображения обоев и построения трёхмерных визуализаций жилых помещений и наложения текстур;

3) спроектировать интерфейс приложения;

4) разработать и провести отладку приложения;

5) протестировать работу приложения.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

При проведении чистовой отделки жилого помещения, необходимо иметь возможность оценить количество необходимого материала и визуализировать возможный результат отделки еще на этапе планирования. В связи с этим было разработано множество мобильных приложений, которые способны выполнять данные функции, однако каждое из них имеет свои недостатки. Необходимо проанализировать рынок мобильных приложений, обладающих функциями, сходными с теми, что необходимо получить в результате разработки. В качестве основных критериев для сравнения выберем следующие:

– построение трёхмерной модели помещения;

– получение текстуры обоев с помощью фото;

– заполнение модели текстурами в соответствии с размерами и узором обоев;

– возможность сравнения вариантов;

– расчёт количества материалов;

– доступность широкому кругу пользователей без оплаты и авторизации;

– независимость приложения от конкретных поставщиков или производителей отделочных материалов.

1.1 Обзор и сравнение аналогов

На данный момент представлено множество приложений, позволяющих решить часть из поставленных задач, однако, они не предполагают возможности решения всех задач одновременно. Рассмотрим из всех вариантов, наиболее подходящие к заданным требованиям:

– planner5D;

– homestyler;

– строительные калькуляторы от двух разных разработчиков (разработчики: Will Maze, CodART.PRO).

Planner 5D приложение, которое позволяет создать дизайн интерьера для дома в трёхмерном пространстве. Дизайн комнат – инструмент, позволяющий настроить планировку с необходимыми размерами [2].

Приложение позволяет без ограничений выбирать и создавать дизайн этажа и комнат, сравнивать разные варианты дизайна и добавлять элементы интерьера.

Доступно широкому кругу лиц, так как распространяется бесплатно без обязательной авторизации, однако имеет внутренние покупки.

В приложении доступен каталог мебели с предметами для использования в дизайне, без возможности редактирования.

Homestyler – это приложение, которое позволяет планировать пространство, создавать дизайн интерьера, выполнять отделку, расставлять мебель и проводить ремонт дома или квартиры. Используя онлайн-программу планировщика этажей, можно выбрать необходимую мебель, разместить её, чтобы реализовать дизайн пространства и сделать украшение интерьера. Доступен трёхмерный рендеринг помещений, рендеринг реальной визуальной панорамы и заполнение моделей помещений текстурами обоев из каталога [3]. Также имеется возможность сравнивать разные варианты дизайна.

Доступно широкому кругу лиц, так как распространяется бесплатно, однако имеет внутренние покупки и обязательную авторизацию.

Имеет собственный каталог мебели с предметами интерьера, без возможности редактирования данного.

Строительный калькулятор от разработчика Will Maze – это приложение, которое может рассчитать количество и стоимость материалов для строительства и ремонта, а также производить расчёты, которые связаны с формой помещения [4].

Приложение бесплатное, без обязательной авторизации, однако имеет внутренние покупки.

Строительный калькулятор от разработчика CodART.PRO – это мобильное приложение, способное автоматически определять необходимое количество материалов и рассчитывать их стоимость для процессов отделки или

строительства. Кроме того, приложение обладает функциональностью по конвертированию единиц измерения и вычислению площадей различных геометрических фигур [5].

Приложение бесплатное, без обязательной авторизации, однако имеет внутренние покупки.

В таблице 1 представлен сравнительный анализ аналогов разрабатываемого приложения.

Таблица 1 – Сравнительный анализ аналогов разрабатываемого приложения по некоторым критериям.

Название приложения	**Построение трёхмерной модели**	**Получение текстуры обоев**	**Заполнение модели текстурами**	**Возможность сравнения вариантов**
Planner5D	+	-	+	+
Homestyler	+	-	+	+
Строительный калькулятор (разработчик: Will Maze)	-	-	-	-
Строительный калькулятор (разработчик: CodART.PRO)	-	-	-	-

Продолжение таблицы 1

Название приложения	**Расчёт количества материалов**	**Доступность широкому кругу пользователей**	**Независимость приложения от поставщиков**
Planner5D	-	+	-
Homestyler	-	+	-
Строительный калькулятор (разработчик: Will Maze)	+	+	-
Строительный калькулятор (разработчик: CodART.PRO)	+	+	-

Исходя из данных, приведённых в таблице 1, можно прийти к выводу, что проблемы, которые возникают у людей в процессе чистовой отделки помещения можно решить, разработав мобильное приложение, которое объединило бы в себе функционал одних аналогов с недостающим им функционалом других.

1.2 Определение требований

В начале работы были определены функциональные требования, которые описывают основной функционал приложения. На рисунке 1 изображена диаграмма вариантов использования мобильного приложения. Пользователь является ключевым актёром, взаимодействующим с мобильным приложением. Пользователем является человек, который может быть неспециалистом в сфере отделки жилых помещений.

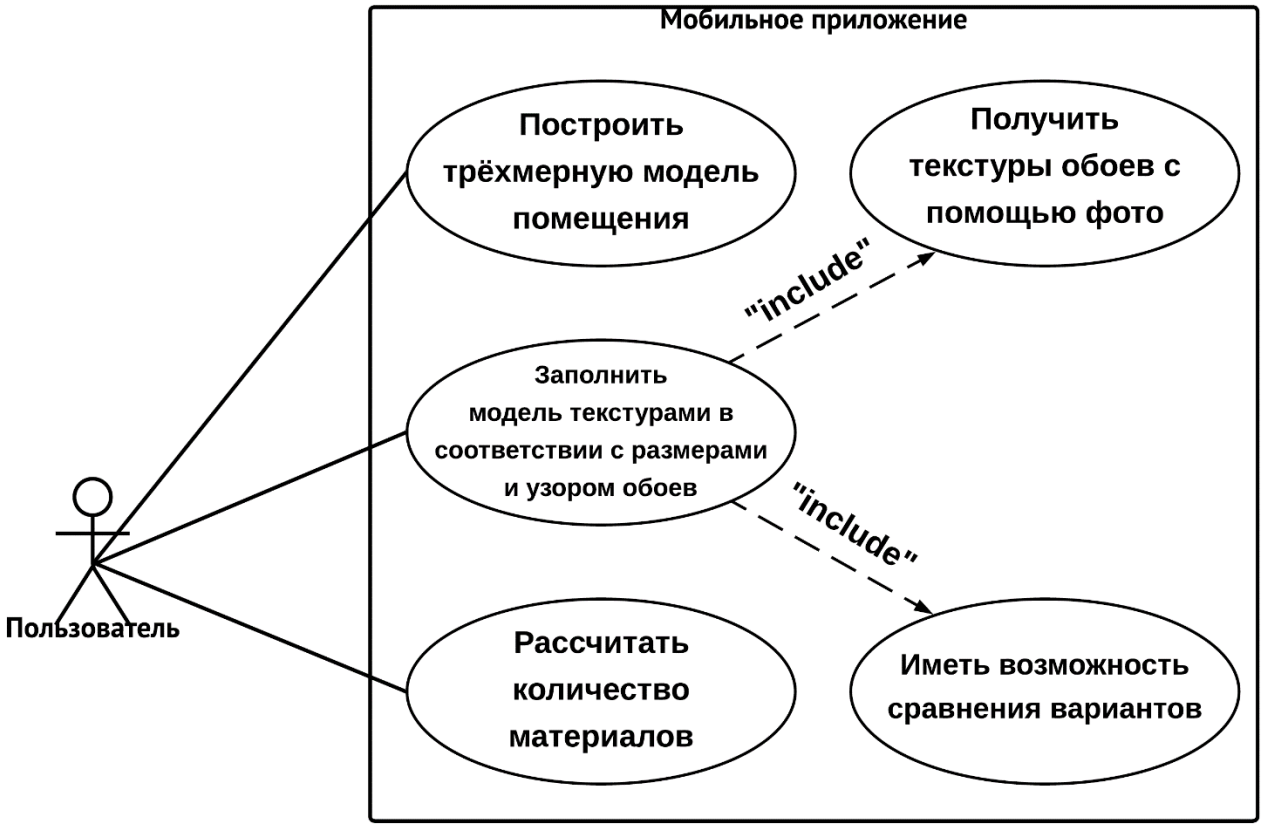

Рисунок 1 – Диаграмма вариантов использования мобильного приложения

Рассмотрим прецеденты использования разрабатываемого мобильного приложения.

Пользователь может использовать функционал приложения бесплатно и без авторизации. После запуска приложения он получает доступ к функциональности по созданию трёхмерной модели помещения, которая будет использоваться для проведения процесса отделки.

На странице редактора трёхмерной модели помещения появляется возможность применить фотографические текстуры обоев для заполнения модели в соответствии с их размерами и узорами, а также возможность сравнения различных текстур.

Также пользователь имеет возможность осуществить расчёт требуемого объема материалов.

В дальнейшем были определены нефункциональные требования, которые описывают дополнительные характеристики приложения.

К нефункциональным требованиям можно отнести:

– доступность широкому кругу пользователей без оплаты и авторизации;

– независимость приложения от конкретных поставщиков или производителей отделочных материалов.

Первое требование подразумевает доступность людям, которые могут не являться специалистами в области отделки помещений, не имеющим возможности оплаты данного функционала и тем, для кого авторизация в приложение может оказаться трудным шагом для получения необходимого функционала.

Второе требование подразумевает возможность приложения, с помощью технологии сканирования обоев через фото, быть независимым от конкретных поставщиков или производителей отделочных материалов.

1.3 Анализ основных технологических решений

Для разработки приложения была выбрана операционная система (ОС) Android, так как она обладает определёнными преимуществами перед IOS:

– большое количество пользователей;

– большое сообщество разработчиков;

– для работы подойдёт любой компьютер;

– исходный код Android открыт для всех;

– публикация приложений в Google Play происходит практически мгновенно, обычно без участия модераторов – не нужно долго ждать подтверждения;

– небольшая плата за размещение приложений в Google Play в сравнении с App Store ($25 один раз, при регистрации, вместо $99 ежегодно) [6].

Статистика сайта statcounter.com (рисунок 2) позволяет сделать вывод о том, что доля ОС Android на рынке мобильных операционных систем по всему миру в феврале составляет 69,13%, а в то же время доля ОС IOS на рынке в феврале составляет 30,11% [7].

Это означает, что ОС Android пользуется более высоким спросом у людей в мире, чем IOS.

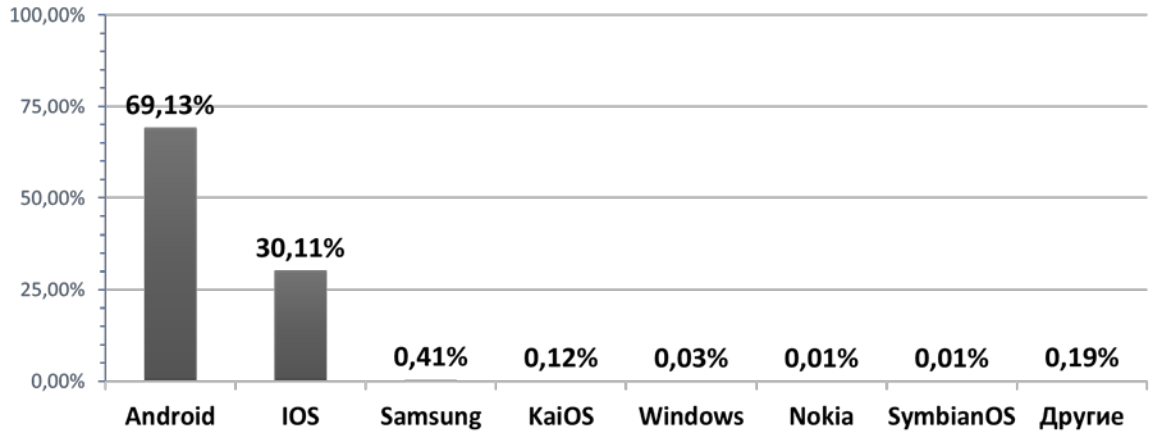

Рисунок 2 – Доля рынка операционных систем по всему миру

Так как в качестве ОС была выбрана Android, соответственно ей рассматривалось использование официальных языков для разработки Java и Kotlin [8].

Они обладают общими чертами:

– являются языками со статической типизацией. Это означает, что проверка типов выполняется во время компиляции;

– являются бесплатными и распространяются как открытое программное обеспечение;

– преобразуют код в байт-код, исполняемый Java Virtual Machine;

– являются интероперабельными. Это означает, что файлы Java и Kotlin могут сосуществовать в одном проекте или пакете JAR;

– являются объектно-ориентированными языками программирования.

Однако также языки имеют определённые различия:

– по скорости Java превосходит Kotlin на 12–15 % для чистых сборок. Однако в случае частичных сборок с включённой инкрементной компиляцией Kotlin компилируется так же, как и Java;

– код, написанный на Kotlin, намного компактнее по сравнению с Java – на 30–40 %. Таким образом, в теории размер приложений может уменьшиться на треть;

– безопасная работа с обнуляемыми переменными. Kotlin защищён от NullPointerException. Этот тип ошибки является самой частой причиной сбоев приложений из Google Play [9].

С учетом сравнения этих языков программирования, для разработки приложения был выбран язык Kotlin. Он является официальным языком разработки для платформы Android, также как и Java, и подходит для начала разработки на этой платформе. Более компактный синтаксис Kotlin позволяет сократить размер приложений на треть по сравнению с Java, что в свою очередь способствует уменьшению времени разработки приложения. Также в качестве среды разработки была выбрана такая среда, как Android Studio [10], так как она имеет весь необходимый функционал, который позволяет осуществить разработку мобильного приложения.

Выводы по разделу один

В результате проведённого анализа аналогов были выявлены основные недостатки существующих приложений. Проанализированы также различные ОС и инструменты для разработки. Основные преимущества приложения, которым оно должно обладать в отличии от аналогов:

– отмена встроенных покупок (приложение должно быть максимально доступным широкому кругу пользователей);

– отмена обязательной авторизации (также позволяет использовать приложение более широкому кругу лиц);

– построение трёхмерной моделей помещения и системы расчёта количества необходимых материалов (так как аналоги обладают похожими свойствами

обособленно друг от друга, необходимо совместить эти функции в одном приложении).

Для реализации разработки приложения была выбрана ОС Android, язык программирования Kotlin и интегрированная среда разработки Android Studio. Определение Android в качестве операционной системы обусловлено его преимуществами перед IOS. Kotlin был выбран в качестве языка программирования, так как является официальным языком разработки для Android и обладает компактностью и легкостью в сравнении с Java. Android Studio была выбрана в качестве среды разработки, так как обладает необходимым для разработки мобильного приложения функционалом.

2 РАЗРАБОТКА АЛГОРИТМОВ ОБРАБОТКИ ИЗОБРАЖЕНИЯ ОБОЕВ И ПОСТРОЕНИЯ ТРЁХМЕРНЫХ ВИЗУАЛИЗАЦИЙ ЖИЛЫХ ПОМЕЩЕНИЙ С НАЛОЖЕНИЕМ ТЕКСТУР

2.1 Разработка алгоритма обработки изображения обоев

Одним из ключевых аспектов при создании визуализации является обработка фотографических изображений обоев. Именно от качества обработки и наложения текстур зависит реалистичность и привлекательность визуализации, а также возможность пользователя оценить, как будет выглядеть интерьер после окончания ремонта и отделки.

Для того, чтобы произвести обработку фотографий, необходимо разработать алгоритм.

На рисунке 3 представлен оптимальный алгоритм, который позволит решить поставленную задачу обработки фотографического изображения обоев.

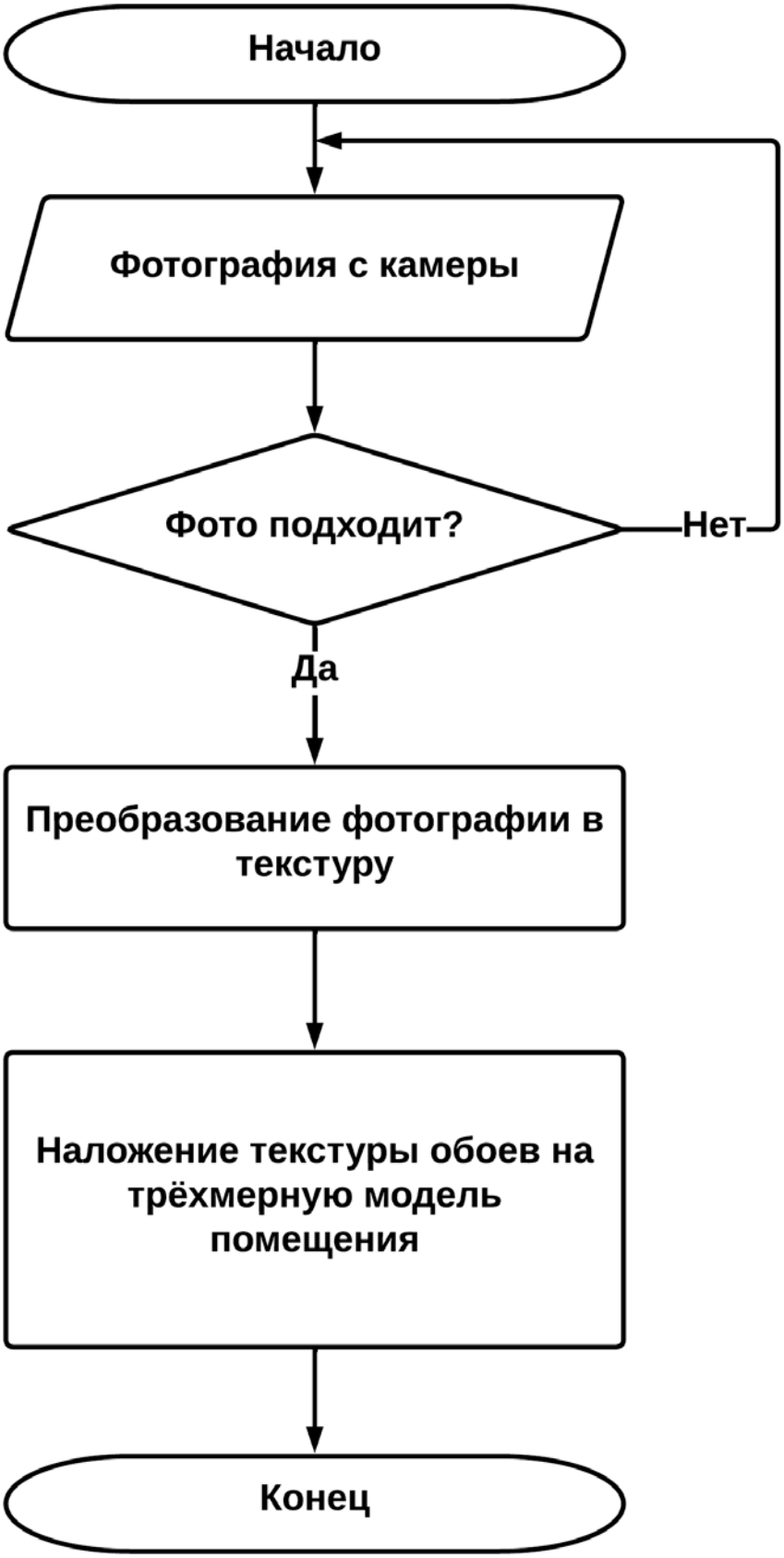

Рисунок 3 – Алгоритм обработки фотографического изображения обоев

Первый шаг данного алгоритма – это получения фотографии обоев с использованием камеры устройства.

Вторым шагом является оценка соответствия фотографии для использования её в качестве текстуры обоев. Главной целью этого этапа является получение фотографии, которая максимально удовлетворяет индивидуальным предпочтениям пользователя.

Третьим шагом при условии того, что фотография удовлетворяет требованиям пользователя, является преобразование фотографии в текстуру. Однако если фотография не удовлетворяет требованиям пользователя, то происходит возвращение к первому шагу алгоритма. Этот этап предоставляет возможность повторить процесс фотографирования и оценки изображения до достижения желаемого качества.

Четвёртый шаг подразумевает наложение данной текстуры на трёхмерную модель помещения.

Алгоритм завершается после четвёртого шага. После успешного выполнения алгоритма обработки фотографического изображения обоев, пользователь получает трёхмерную модель помещения, на которую нанесена текстура обоев, полученная в процессе выполнения алгоритма.

Для реализации описанного алгоритма обработки фотографий в приложении был создан специальный компонент с приватным идентификатором “cameraLauncher”. Данный компонент осуществляет взаимодействие с камерой устройства и запускает процесс фотографирования. После получения снимка, который удовлетворяет требованиям пользователя, происходит обновление текстур обоев на трёхмерной модели помещения, а также перерисовка сцены. В случае, если полученная фотография не соответствует требованиям, пользователь возвращается либо в редактор модели помещения, либо в интерфейс камеры для того, чтобы сделать фотографию снова.

Алгоритм реализован с использованием функционала, предоставляемого Android-фреймворком. В частности, для запуска камеры и получения фотографии используется класс “ActivityResultContracts.TakePicturePreview()”. После

получения результата фотографирования, происходит проверка на наличие снимка “(result != null)”. В случае, если снимок получен успешно, вызывается метод “updateWallTextureBitmap()”, который обновляет текстуры обоев на трёхмерной модели помещения, передавая ему полученное изображение. Затем вызывается метод “requestRender()”, который перерисовывает сцену с обновленными текстурами. В случае, если снимок не был получен “(result = null)”, алгоритм завершается без дальнейших действий.

Предварительно проверяется наличие разрешения на использование камеры устройства. Если разрешение не было предоставлено, запрашивается разрешение с помощью метода “requestPermissions()”. После получения разрешения или если разрешение уже было предоставлено, происходит запуск камеры с помощью метода “launch()” компонента “cameraLauncher”.

Реализация данных методов приведена в приложениях А и Б.

Таким образом, реализация данного алгоритма обеспечивает возможность фотографирования, обновление текстур обоев на трёхмерной модели помещения и перерисовку сцены в соответствии с полученными изображениями, а также предусматривает возврат пользователя в редактор модели помещения в случае неудовлетворительного результата фотографирования.

2.2 Разработка алгоритма построения трёхмерных визуализаций жилых помещений с наложением текстур

Для создания трёхмерной визуализации жилых помещений с наложением текстур был разработан соответствующий алгоритм, который включает ряд этапов и проверок для достижения желаемого результата.

На рисунке 4 представлен оптимальный алгоритм, который позволит решить поставленную задачу построения трёхмерных визуализаций жилых помещений с наложением текстур.

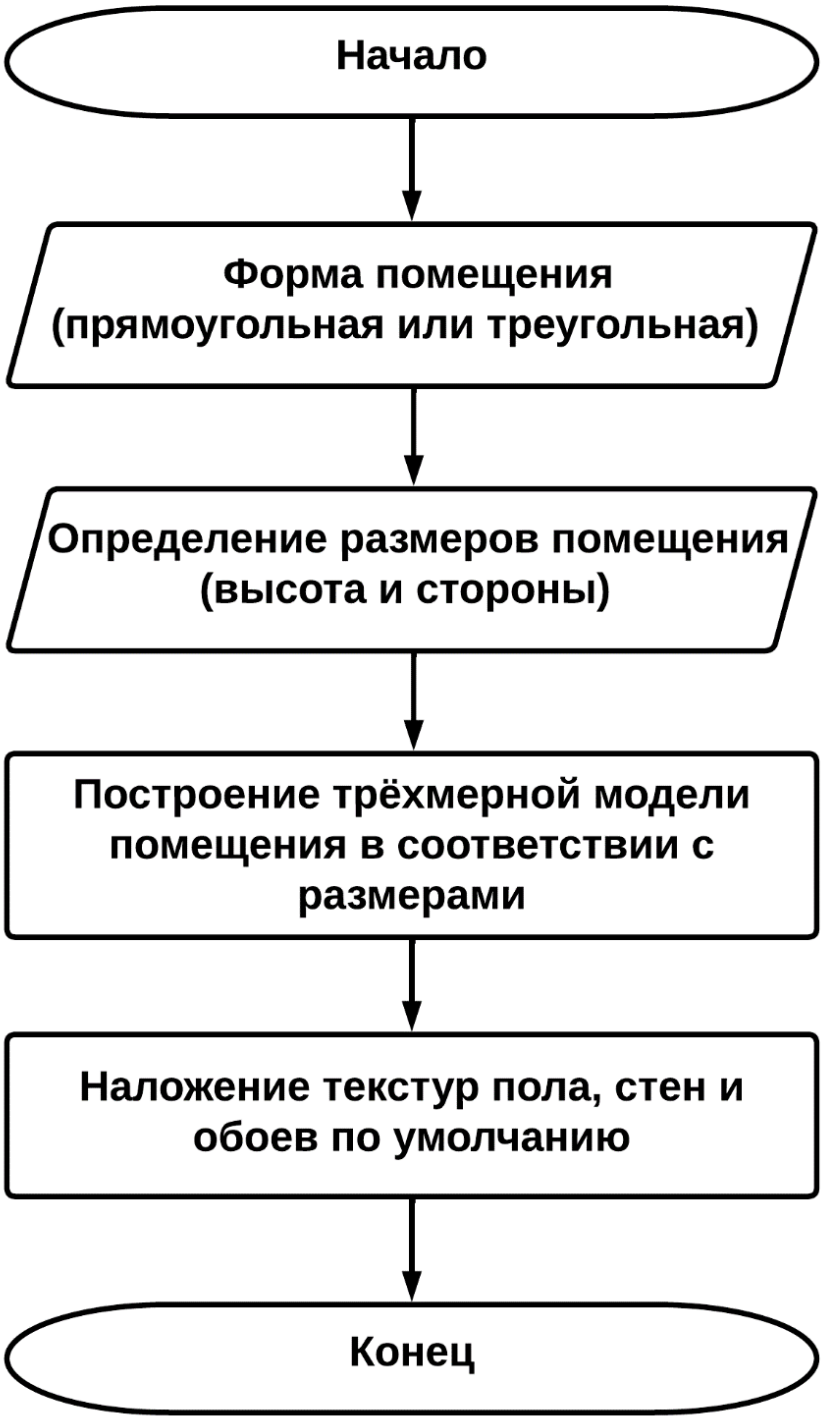

Рисунок 4 – Алгоритм построения трёхмерных визуализаций жилых помещений с наложением текстур

Первый шаг данного алгоритма – это получение информации о форме помещения, которая на данный момент ограничивается прямоугольной и треугольной формами.

Вторым шагом требуется ввести соответствующие параметры, такие как размеры сторон и высота помещения. Эти данные необходимы для точного построения трёхмерной модели, которая отображает геометрические особенности и размеры указанного помещения.

Третьим шагом является построение трёхмерной модели помещения, которая представляет собой графическое представление его структуры в соответствии с размерами помещения. Этот шаг определяет границы и контуры поверхностей трёхмерной модели.

Четвертый шаг – это наложение текстур пола, стен и обоев по умолчанию. Этот шаг позволяет наложить заранее подготовленные текстуры на трёхмерную модель помещения.

В результате применения алгоритма получается трёхмерная визуализация жилого помещения, где текстуры обоев отображаются на соответствующих поверхностях недостаточно корректным образом, так как растягиваются на всю площадь поверхности, а не представляются в виде отдельных частей на поверхности, которые совместно могут создать корректное визуальное представление об обоях. Данный недостаток в разработке связан с ограничением времени, которое выделено на выпускную квалификационную работу, однако в дальнейшем он может быть исправлен при корректировке методов, которые накладывают текстуры на модель помещения. Разработанный алгоритм служит инструментом, способствующим принятию решений в отношении отделки и внутреннего оформления помещений, упрощая процесс проектирования и визуализации.

Реализация данного алгоритма основана на следующих особенностях.

В классе “Editor3Activity.kt” происходит инициализация параметров помещения, таких как длина стороны A, длина стороны B, высота помещения и флаг, указывающий на треугольную форму. Если форма треугольная, то также определяется длина стороны C. Создается объект “room3D” класса “Room3D.kt”, который содержит информацию о параметрах помещения.

В методе “onSurfaceCreated” класса “MyRenderer.kt” устанавливаются параметры отрисовки, включая очистку цветового буфера и включение теста глубины. Затем вызывается метод “createAndUseProgram()”, который создает программу OpenGL (OpenGL – это название спецификации, описывающей поведение системы рендеринга на основе растеризации. Он определяет API (Application Programming Interface – интерфейс, который позволяет связывать разные приложения), с помощью которого клиентское приложение может управлять этой системой [11]) и связывает ее с контекстом. Метод “locations()” определяет местоположение атрибутов и uniform-переменных в шейдере. Затем вызывается метод “prepareData()”, который подготавливает данные для отрисовки. Создается матрица проекции и матрица вида, которые передаются в шейдер через uniform-переменную “u_Matrix”.

Метод “onSurfaceChanged” вызывается при изменении размера поверхности отрисовки. В этом методе определяются параметры области отображения и создается матрица проекции с учетом новых размеров.

Метод “onDrawFrame” отвечает за основной цикл отрисовки сцены. Внутри него вызывается метод “createViewMatrix()”, который обновляет матрицу вида. Затем вызывается метод “bindMatrix()”, который передает матрицу проекции и матрицу вида в шейдер. После этого происходит очистка цветового буфера и буфера глубины. Затем вызываются методы “glDrawArrays”, которые обрисовывают каждую грань помещения с использованием соответствующих текстур. Параметры для каждой грани и текстуры определяются на основе данных из объекта `room3D`.

Класс “MyRenderer” также содержит методы “updateWallTexture’ и “updateWallTextureBitmap”, которые обновляют текстуру стен помещения.

Реализация данных методов представлена в приложениях А и Б.

Таким образом, реализация алгоритма включает инициализацию параметров помещения, создание программы OpenGL, передачу данных в шейдеры, установку матрицы проекции и матрицы вида, а также последовательную отрисовку каждой грани помещения с использованием соответствующих текстур. Это позволяет

создать трёхмерное представление помещения с прямоугольной или треугольной формой и применить текстуры стен, пола и обоев.

Выводы по разделу два

После разработки алгоритмов обработки фотографического изображения обоев и построения трёхмерных визуализаций жилых помещений с наложением текстур, получены важные компоненты для разработки приложения, которое должно обладать данным функционалом.

Такое приложение предоставляет возможность пользователям получить представление о том, как изменится интерьер во время декорирования. На данный момент это обеспечивается недостаточно корректным образом, однако может быть исправлено в будущих версиях приложения.

3 ПРОЕКТИРОВАНИЕ ИНТЕРФЕЙСА ПРИЛОЖЕНИЯ

В процессе проектирования интерфейса приложения были выполнены следующие этапы. Вначале была разработана концепция приложения, на основе которой была создана карта пути пользователя.

При открытии приложения пользователю предоставляется возможность либо создать новую трёхмерную модель жилого помещения, либо загрузить ранее сохраненный проект. Кроме того, каждое окно приложения обеспечивает функциональность возврата к предыдущему окну.

После загрузки ранее сохраненного проекта пользователь попадает в редактор трёхмерной модели помещения, который позволяет добавлять различные объекты в модель, сохранять текущую конфигурацию и производить расчет материалов, необходимых для отделки помещения.

Добавление объектов включает в себя установку дверей, окон и обоев. При добавлении дверей и окон пользователь указывает их размеры для последующих расчетов. При добавлении обоев предлагается выбор: создание новых с помощью камеры или загрузка предварительно сохраненных вариантов.

Сохранение текущей конфигурации позволяет пользователю сохранить проект для последующей загрузки и работы над ним.

Расчет материалов предоставляет возможность определить стоимость и количество необходимых материалов на основе введенных пользователем цен.

При создании новой трёхмерной модели пользователю предлагается выбор двухмерной формы помещения. Двухмерная форма может быть прямоугольной или треугольной. При выборе формы пользователю предлагается указать параметры размеров помещения, такие как высота и длина сторон.

После выбора формы помещения и задания параметров пользователь переходит в редактор трёхмерной модели, где доступны те же функции, что и при загрузке готового проекта.

На рисунке 5 представлена карта пути пользователя, иллюстрирующая последовательность действий пользователя в приложении.

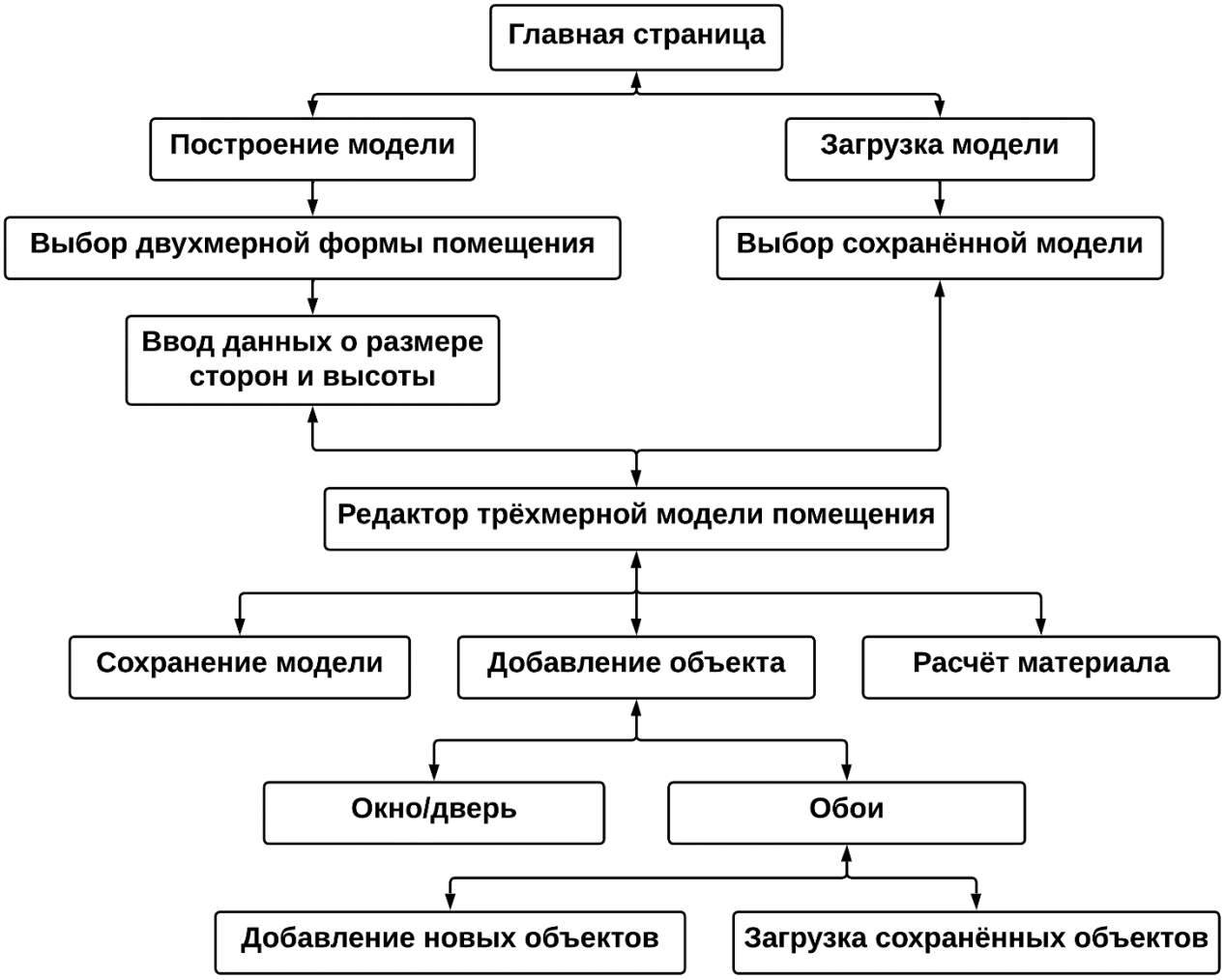

Рисунок 5 – Карта пути пользователя

Выводы по разделу три

В данном разделе работы были выполнены этапы проектирования интерфейса мобильного приложения для отделки жилых помещений. Начальным этапом была разработана концепция приложения, определяющая его основные функциональные возможности. На основе этой концепции была создана карта пути пользователя, которая описывает последовательность действий, доступных пользователю при работе с приложением.

4 РАЗРАБОТКА И ОТЛАДКА ПРИЛОЖЕНИЯ

4.1 Разработка приложения

Процесс разработки приложения был разделён на определённые этапы.

Первым этапом стало определение архитектуры приложения, которая обеспечивает эффективное взаимодействие с пользователем. Процедурная архитектура, используемая в данном приложении, представляет собой простую структуру, где логика приложения организована в процедурном стиле без явного разделения на модули или компоненты. В приложении отсутствуют принятые архитектурные паттерны, такие как MVVM (Model-View-ViewModel) или MVP (Model-View-Presenter), что может привести к сложностям в поддержке и развитии приложения в долгосрочной перспективе. Для улучшения поддержки, масштабирования и возможности изменения пользовательского интерфейса независимо от логики приложения, рекомендуется внести изменения в архитектуру, применив подход Model-View-Controller (MVC) или Model-View-Presenter (MVP). В предлагаемой архитектурной модели, модуль представления будет ответственен за отображение элементов пользовательского интерфейса и обработку пользовательских событий. Модуль логики будет отвечать за обработку бизнес-логики и взаимодействие с моделью данных. Модель данных будет представлять собой набор классов, описывающих данные и операции над ними.

В данном разделе представлен процесс разработки приложения в среде Android Studio, выбранной ранее в качестве основной платформы разработки. Android Studio предоставляет необходимый функционал для разработки мобильных приложений на платформе Android с использованием языка Kotlin. Выбор Kotlin в качестве основного языка программирования позволил сократить объем кода и снизить вес приложения.

В процессе разработки приложения возникли сложности в реализации функциональности загрузки готовых проектов и корректного использования обоев, а также в построении объектов, таких как окна или двери, на трёхмерной модели

помещения. Решение этих задач требует учета множества технических аспектов и обеспечения реалистичной визуализации пространства. В рамках выпускной квалификационной работы эти задачи оказались сложными и не были полностью реализованы. Однако, эти аспекты представляют перспективное направление для дальнейшего развития приложения, где можно будет более детально изучить технические особенности и разработать более продвинутые алгоритмы для обработки таких элементов, чтобы обеспечить более полную функциональность и реалистичность приложения.

В рамках начального этапа разработки был реализован класс главной активности мобильного приложения, а также соответствующий макет, который объединяет в себе функционал загрузки существующего проекта и создания нового проекта. Пользователю предоставляется возможность выбора одного из двух указанных вариантов действий, что представлено на рисунке 6, а также проходя по любому пути пользователь имеет возможность вернуться назад.

Рисунок 6 – Главный экран приложения

При загрузке проекта пользователь сталкивается с пустым экраном, лишенным полного функционала, и предоставляется возможность вернуться обратно (рисунок 7). Трудности, с которыми пришлось столкнуться в процессе разработки, препятствовали полной реализации функционала данного экрана.

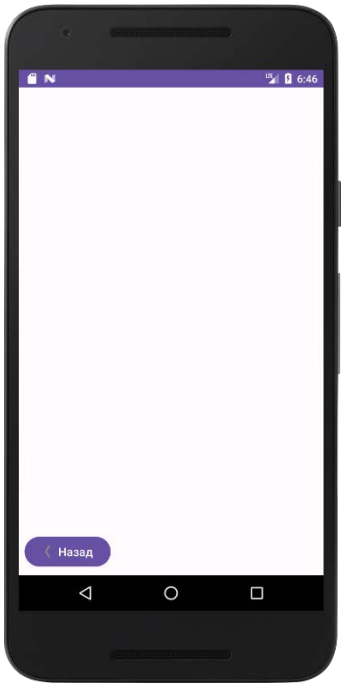

Рисунок 7 – Экран загрузки готового проекта

При создании нового проекта пользователь перенаправляется на экран выбора формы помещения, предоставляющий возможность выбора между прямоугольной и треугольной формами (рисунок 8).

Рисунок 8 – Выбор формы помещения

После выбора формы, открывается диалоговое окно, где пользователю предлагается ввести данные о размерах сторон помещения соответствующей фигуры и высоте помещения, что представлено на рисунках 9 и 10.

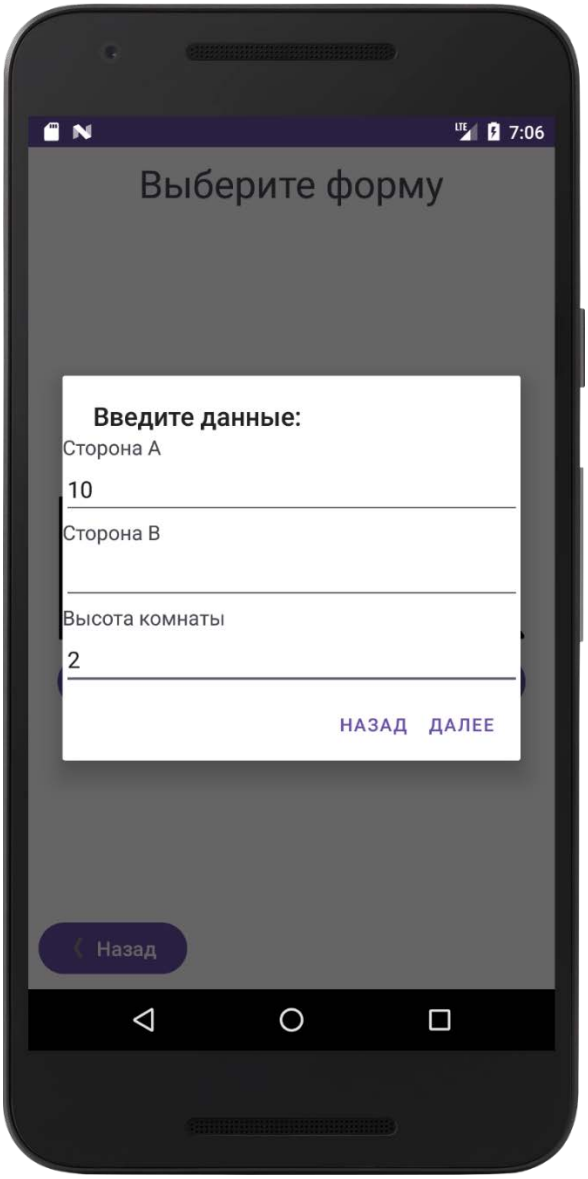

Рисунок 9 – Ввод данных для прямоугольника

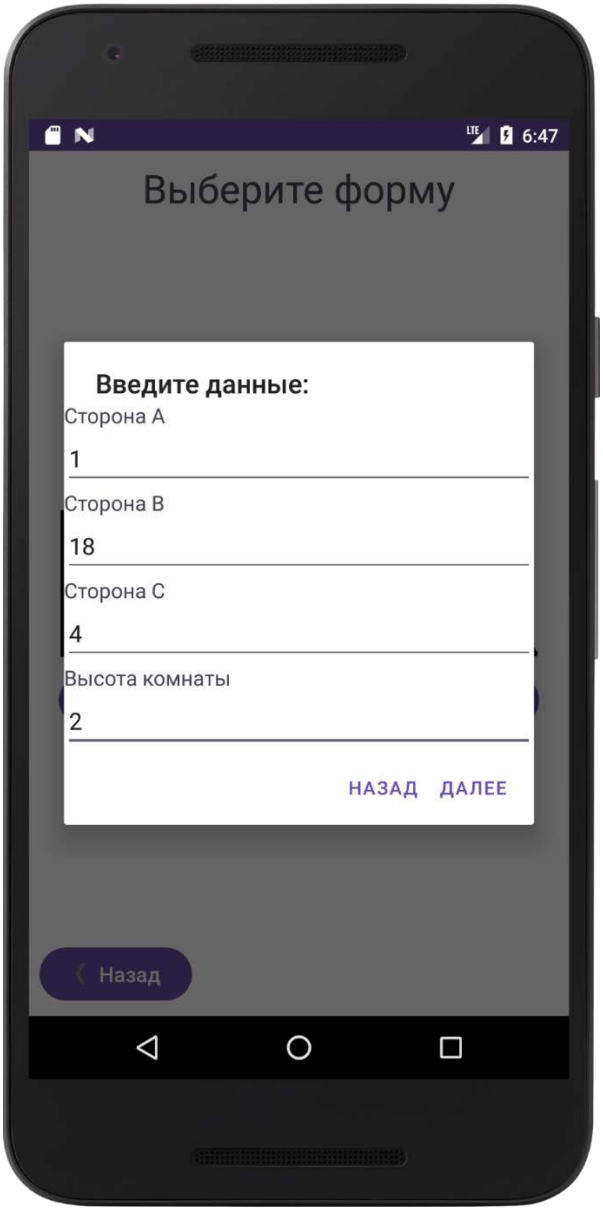

Рисунок 10 – Ввод данных для треугольника

После выбора пользователем формы помещения и ввода соответствующих размеров, система направляет пользователя в специальное окно, где реализован функционал для построения трёхмерной визуализации жилого пространства (рисунок 11). В настоящий момент приложение уже предоставляет возможность добавления обоев на стены с помощью кнопок, которые предлагают выбрать обои, которые загружены по умолчанию, загрузить обои из галереи устройства или сделать фотографию и использовать ее в качестве обоев. Кроме того, также реализованы кнопки для добавления окон или дверей, учитывающие площадь стен, которая будет занята этими элементами. Это важно для последующего функционала кнопки “рассчитать материал”, который позволяет определить

необходимое количество материала для отделки помещения. В настоящее время система предоставляет лишь ограниченный функционал для корректного отображения обоев, окон и дверей на трёхмерной модели помещения, и сохранения проекта. Тем не менее, это открывает потенциал для дальнейшего развития и внедрения более полного функционала, который позволит более точно отображать выбранные элементы на трёхмерной модели и сохранять проект с полной информацией о расстановке элементов в помещении.

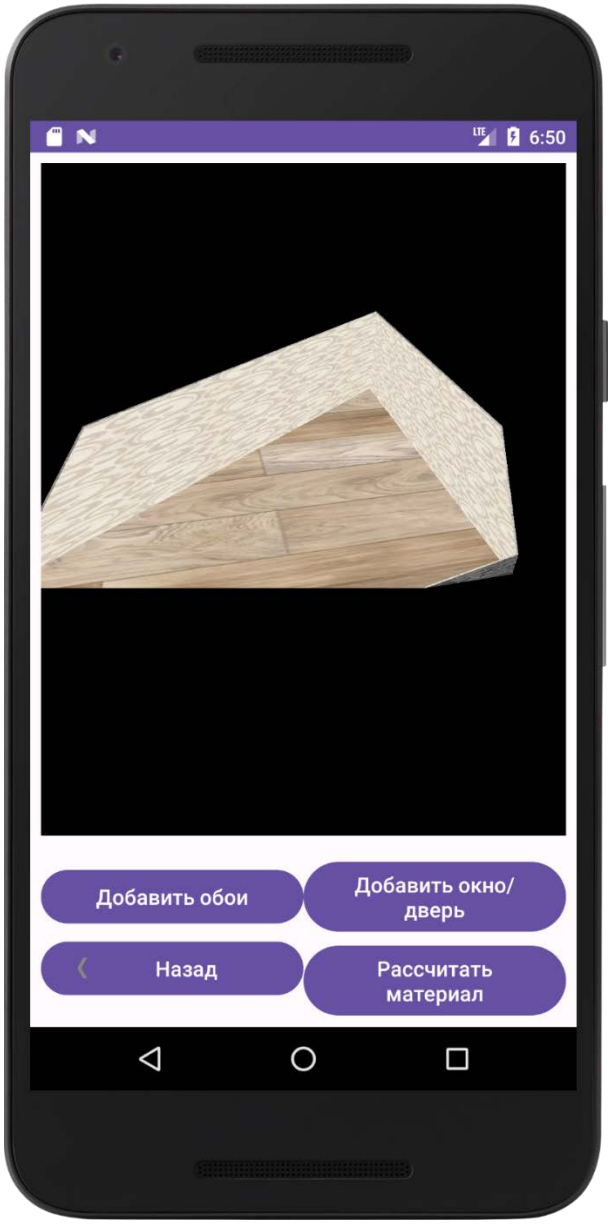

Рисунок 11 – Экран с возможностью добавления объектов и расчёта материала

При активации функционала кнопки «Добавить обои» в пользовательском интерфейсе появляется диалоговое окно, предоставляющее пользователю возможность выбрать один из трех способов загрузки обоев. Первый способ

предусматривает установку обоев по умолчанию, предварительно заданных при разработке приложения. Второй способ позволяет пользователю загрузить собственное изображение обоев из галереи на устройстве. Третий способ предоставляет возможность открыть камеру устройства и сделать фотографию, которая будет использована в качестве обоев. Все эти варианты представлены на рисунке 12.

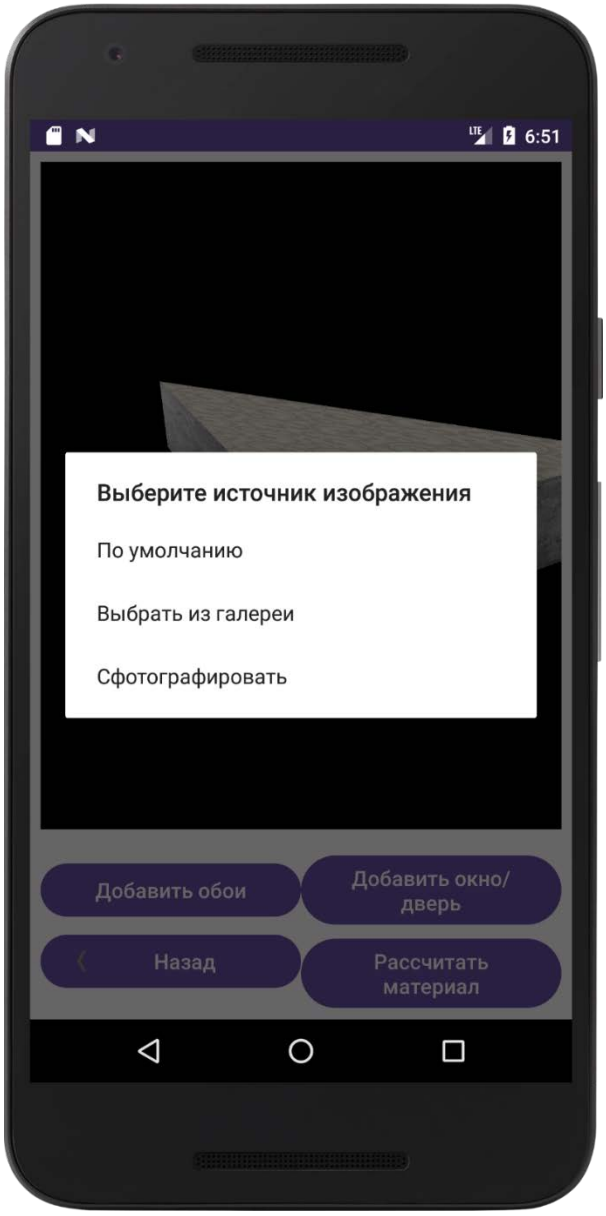

Рисунок 12 – Диалоговое окно для добавления обоев

При активации функционала кнопки «Добавить окно/дверь» в приложении появляется диалоговое окно, предоставляющее пользователю возможность выбрать тип объекта (окно или дверь) и указать соответствующие параметры, такие как высота и ширина, для корректного вычисления общей площади стен внутри

помещения. Этот важный этап проектирования представлен на рисунке 13. В процессе работы с диалоговым окном пользователь может выбрать тип объекта из доступных вариантов и задать его геометрические характеристики, включая длину и ширину. Эти параметры являются необходимыми для дальнейшего вычисления общей площади стен в помещении, что обеспечивает точность и корректность расчетов функционала приложения.

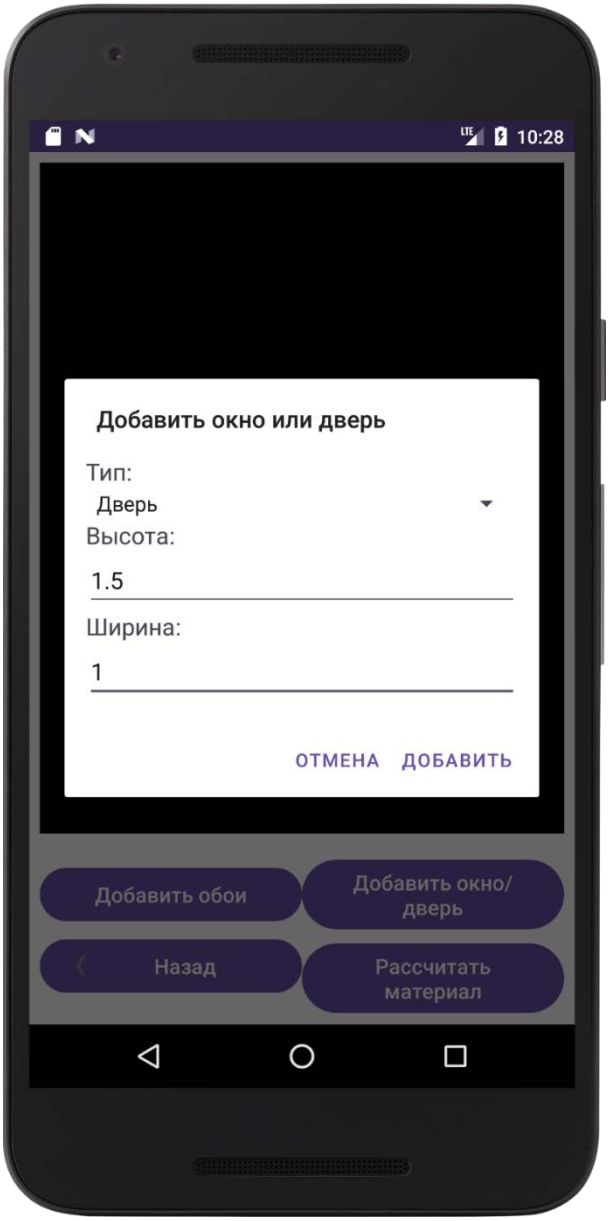

Рисунок 13 – Диалоговое окно для добавления окна/двери

После активации функционала кнопки “Рассчитать материал” запускается диалоговое окно, предоставляющее пользователю возможность выполнить определенные действия с целью оценки стоимости материала. В данном диалоговом окне пользователю предлагается указать цену обоев за рулон, длину и

ширину рулона и осуществить расчет стоимости материала, а также определить необходимое количество материала, требуемое для покрытия стен в помещении. Этот функционал, представлен на рисунке 14, является одним из ключевых элементов разработанного приложения и служит важной составляющей для определения финансовых затрат на отделку помещения. С помощью данного диалогового окна пользователь может ввести информацию о цене материала за единицу площади и осуществить вычисления, необходимые для оценки общей стоимости материала и определения объема, требуемого для покрытия стен в выбранном помещении.

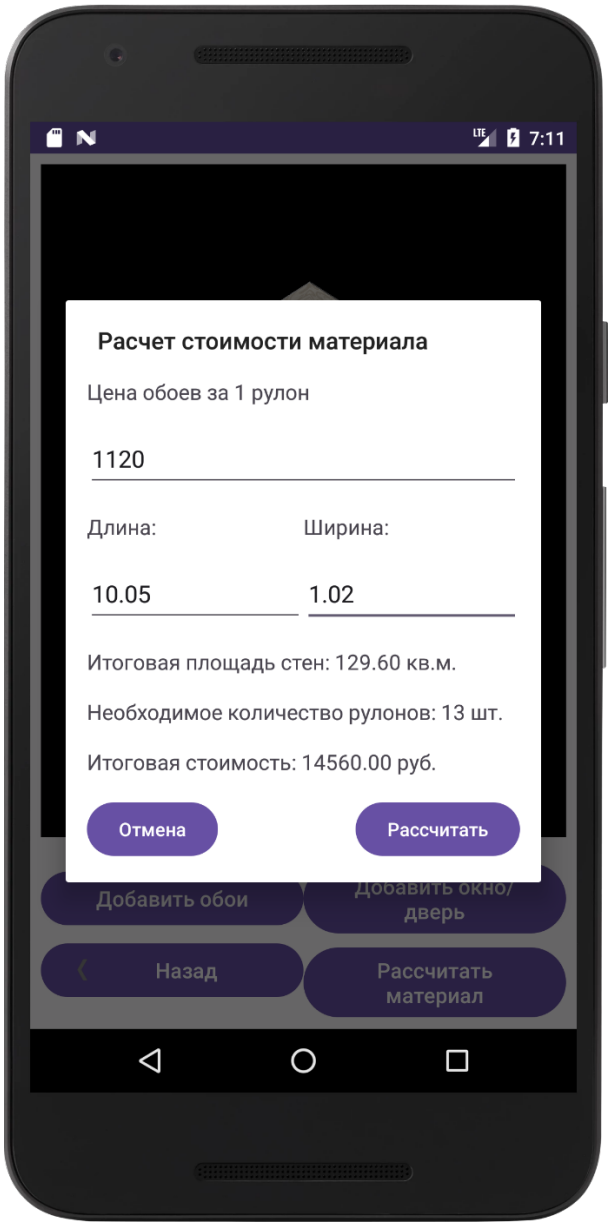

Рисунок 14 – Диалоговое окно для расчёта материала

4.2 Отладка приложения

Для отладки приложения в рамках данного исследования использовалась интегрированная среда разработки Android Studio. Основным инструментом отладки был режим Debug, предоставляемый Android Studio. Этот режим позволял установить точки останова в коде приложения, что позволяло следить за передачей данных между переменными и обнаруживать ошибки, связанные с некорректной инициализацией или неправильной работой приложения.

Дополнительно, для анализа и отслеживания процесса выполнения приложения, в процессе отладки использовался инструмент LogCat, предоставляемый Android Studio. LogCat позволял отслеживать журнал событий приложения, включая вывод информационных сообщений, предупреждений и ошибок. Это было особенно полезно при обнаружении и анализе проблем в процессе выполнения приложения.

Вместе с использованием Android Studio, режима Debug и инструмента LogCat, была применена методология систематической отладки, включающая установку точек останова в критических местах кода, последовательное тестирование функциональности, анализ вывода LogCat и пошаговый анализ данных для обнаружения и исправления ошибок.

Этот подход к отладке приложения позволил эффективно идентифицировать и исправить различные проблемы, связанные с инициализацией переменных и некорректной работой приложения, обеспечивая стабильность и надежность при выполнении функциональности.

Выводы по разделу четыре

Процесс разработки приложения был разделен на определенные этапы, начиная с определения архитектуры приложения, а также выбора Android Studio в качестве основной платформы разработки и Kotlin в качестве основного языка программирования.

В разработке приложения возникли трудности, связанные с реализацией функциональности загрузки проектов и использования обоев, а также добавления

дверей и окон не только в качестве данных, но и в качестве объектов на трёхмерную модель помещения. Эти аспекты представляют сложные задачи, требующие учета технических и эстетических аспектов, и являются перспективным направлением для дальнейшего развития приложения.

В рамках начальной фазы разработки был реализован класс главной активности приложения, предоставляющая пользователю функционал загрузки существующего проекта и создания нового проекта. Однако полный функционал не был реализован из-за ограниченности временных ресурсов.

Для отладки приложения использовалась интегрированная среда разработки Android Studio с режимом Debug и инструментом LogCat. Этот подход позволил эффективно идентифицировать и исправить ошибки, связанные с инициализацией переменных и некорректной работой приложения.

В целом, разработка приложения приостановлена из-за ограниченности временных ресурсов, однако были достигнуты некоторые промежуточные результаты и применены методы отладки для обеспечения стабильности и надежности приложения.

5 ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ

В данной главе рассматривается процесс тестирования разработанного приложения, который направлен на проверку его работоспособности, соответствия требованиям и выявление возможных проблем и дефектов. Тестирование приложения является важным этапом разработки, позволяющим убедиться в качестве программного продукта перед его внедрением в реальные условия использования. В данной главе представлены результаты проведенного тестирования, анализ полученных данных и выводы, которые могут быть использованы для улучшения приложения и разработки рекомендаций для дальнейшего развития.

5.1 Планирование тестирования приложения

Начальным этапом планирования тестирования является четкое определение целей и задач, которые должны быть достигнуты в процессе тестирования приложения. Цель тестирования заключается в проверке функциональности, надежности и соответствия приложения требованиям. Задачи тестирования включают в себя идентификацию потенциальных дефектов, проверку корректности работы приложения в различных сценариях использования, а также оценку его производительности и безопасности.

После определения целей и задач тестирования необходимо выбрать наиболее подходящие методы и подходы, которые будут применяться в процессе тестирования данного приложения. Методы тестирования могут варьироваться в зависимости от типа приложения, его особенностей и требований. Некоторые из распространенных методов включают модульное тестирование, интеграционное тестирование, системное тестирование.

Модульное тестирование. Тестируется минимально возможный для тестирования компонент, например отдельный класс или функция.

Интеграционное тестирование. Тестируются интерфейсы между компонентами, подсистемами или системами. При наличии резерва времени на данной стадии тестирование ведется итерационно с постепенным подключением последующих подсистем.

Системное тестирование. Тестируется интегрированная система на ее соответствие требованиям [12].

5.2 Проведение тестирования приложения

В начале этапа проведения тестирования подготавливается тестовая среда, включающая необходимые аппаратные и программные компоненты, а также предоставляющая условия, максимально приближенные к реальным, для проведения тестовых сценариев. В качестве тестовой среды используется Android Studio, так как эта среда обладает нужным функционалом.

При выполнении модульного тестирования приложения была осуществлена проверка правильной функциональности определенных методов, включая ввод данных о фигуре в классе активности создания трёхмерной модели помещения “CreateActivity.kt”, который приведён в приложении В. В процессе тестирования были выявлены ошибки, которые не учитывали определенные условия, при которых фигуры не могут существовать (например, отсутствие данных о размерах комнат, невозможность существования треугольника, у которого сумма двух сторон меньше третьей стороны, или использование отрицательных значений). Для исправления этих ошибок было реализовано информационное сообщение пользователю, указывающее на невозможность создания таких фигур при некорректном вводе данных (рисунок 15).

Таким образом, проведение модульного тестирования позволило выявить и устранить ошибки, связанные с некорректным вводом данных о фигуре. Это улучшило общую надежность и функциональность приложения, обеспечивая более точное и корректное взаимодействие с пользователем.

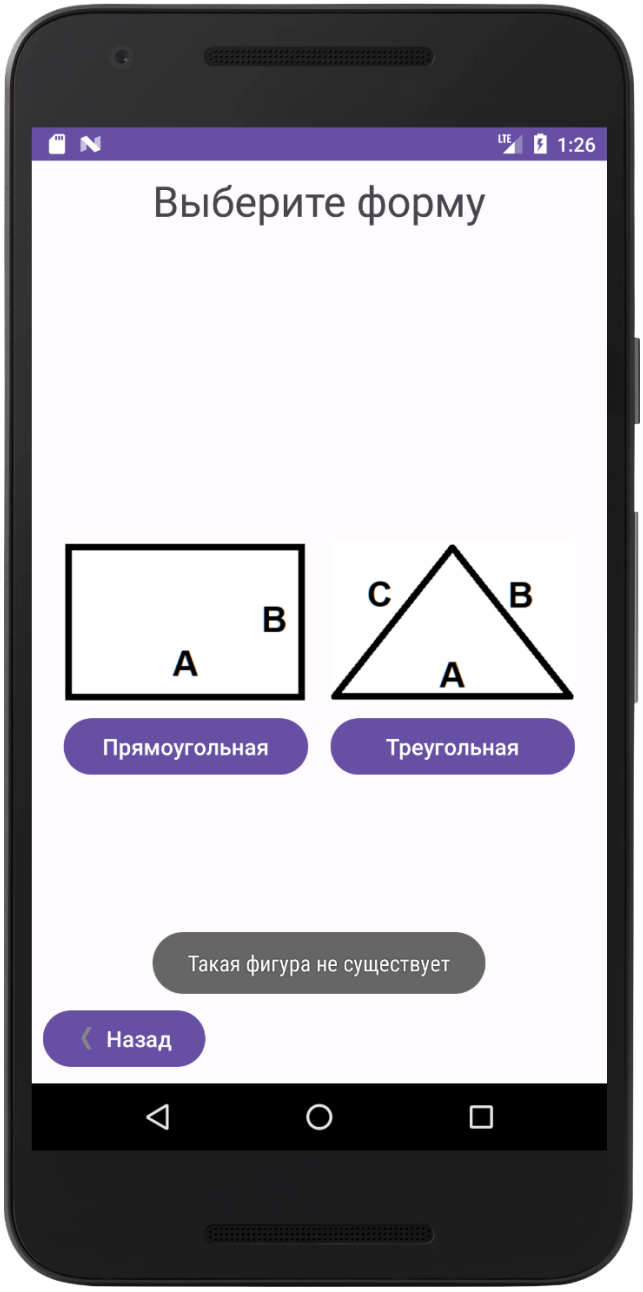

Рисунок 15 – Уведомление о несуществующей фигуре

В рамках интеграционного тестирования приложения была проверена передача данных между классами активностей с использованием механизма Intent. В процессе тестирования было выявлено некорректное передача определенных данных, что приводило к ошибкам, связанным с несоответствием типов данных и вычислительными ошибками. Для исправления данных проблем были разработаны и проведены тесты в режиме отладки (Debug) среды программирования Android Studio.

Тестирование в режиме Debug позволило выявить и устранить ошибки, связанные с передачей данных через Intent, а также обеспечило корректное

соответствие типов данных и надежность вычислений. Таким образом, интеграционное тестирование сыграло важную роль в обеспечении стабильной работы приложения при передаче данных между различными компонентами.

В рамках системного тестирования приложение было проверено на эмуляторах, имитирующих устройства с операционной системой Android версии 7.0 и выше, с уровнем API 24 и выше. В ходе тестирования было установлено, что приложение функционирует корректно, однако при задании больших параметров помещения процесс построения трёхмерной модели и анимации просмотра может потреблять значительные ресурсы и повышать требования к производительности системы для обеспечения стабильной работы приложения.

Однако на физических устройствах не было обнаружено указанных проблем. Исходя из этого, сделано предположение, что такое поведение приложения может быть связано с особенностями эмулятора, который может потреблять значительное количество оперативной памяти компьютера, на котором проводилось тестирование. В результате приложение продемонстрировало стабильную работу без сбоев на физических устройствах.

Это наблюдение позволяет заключить, что приложение работает стабильно и эффективно на реальных устройствах, исключая возможные проблемы, связанные с ресурсами эмулятора и конфигурацией тестирующей системы.

Выводы по разделу пять

В данном разделе представлен процесс тестирования разработанного приложения, направленного на проверку его работоспособности, соответствия требованиям и выявление возможных проблем и дефектов. Были определены цели и задачи тестирования, а также выбраны подходы и методы, включая модульное тестирование, интеграционное тестирование и системное тестирование.

В результате модульного тестирования были выявлены ошибки, связанные с некорректным вводом данных о фигуре, и они были успешно исправлены. Интеграционное тестирование позволило выявить и устранить ошибки, связанные с передачей данных между компонентами приложения. Системное тестирование

проводилось на эмуляторах и физических устройствах, и приложение продемонстрировало стабильную работу на реальных устройствах.

Таким образом, тестирование приложения позволило улучшить его надежность, функциональность и производительность, а также обеспечить корректную передачу данных и стабильную работу на реальных устройствах. Полученные результаты тестирования и анализ данных могут быть использованы для улучшения приложения и разработки рекомендаций для дальнейшего развития.

ЗАКЛЮЧЕНИЕ

В данной выпускной квалификационной работе было разработано приложение для создания трёхмерных визуализаций жилых помещений. Целью работы было создание удобного инструмента, позволяющего пользователям визуализировать и настраивать интерьеры помещений с помощью трёхмерных моделей. В процессе разработки были решены различные технические и функциональные задачи, что привело в частных случаях к некорректной реализации основного функционала приложения, однако может быть доработано в следующих версиях приложения.

В ходе работы был проведен анализ существующих аналогов, что позволило определить основные требования к функциональности и пользовательскому интерфейсу разрабатываемого приложения. Были выбраны необходимые инструменты и технологии для реализации проекта. Также были реализованы ключевые функции, такие как выбор формы и размеров помещения, добавление обоев, окон и дверей на данный момент только в качестве данных, которые учитываются при расчётах необходимого количества материала и его стоимости.

Важной частью работы было тестирование приложения, которое позволило выявить и исправить различные ошибки и проблемы. Были проведены модульные, интеграционные и системные тесты, что обеспечило проверку работоспособности функционала приложения в различных сценариях использования. Тестирование на реальных устройствах и эмуляторах Android позволило подтвердить стабильную работу приложения и выявить некоторые особенности взаимодействия с ресурсами системы.

В результате выполненной работы было создано приложение, которое на данный момент не реализует полностью функционал трёхмерной визуализации жилых помещений. Приложение обладает интуитивно понятным пользовательским интерфейсом и удобными инструментами для настройки интерьера. Благодаря проведенному тестированию были выявлены и исправлены ошибки, что способствует более надежной работе приложения.

Результаты данной работы могут быть использованы в различных сферах, связанных с дизайном интерьера, строительством и ремонтом. Приложение имеет потенциал для дальнейшего развития и расширения функциональности так как не хватило времени реализовать сохранение и загрузку проектов, корректное отображение текстур обоев и добавление окон или дверей на трёхмерную модель помещения.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1 Просто о сложном: внутренняя отделка коммерческих помещений – ООО «ЗЛАТА». – Текст : электронный // ООО «ЗЛАТА» : [сайт]. – 2018. – URL: https://zlataooo.ru/news/otdelochnye-raboty/vnutrennyaya-otdelka-pomeshchenij-normativy-i-osobennosti/ (дата обращения 03.02.2023).

2 Приложения в Google Play – Planner 5D – дизайн интерьера. – Текст : электронный // Google Play : [сайт]. – 2023. – URL: https://play.google.com/store/apps/details?id=com.planner5d.planner5d (дата обращения: 10.02.2023).

3 Приложения в Google Play – Homestyler Дизайн интерьера. – Текст : электронный // Google Play : [сайт]. – 2023. – URL: https://play.google.com/store/apps/details?id=com.autodesk.homestyler (дата обращения: 10.02.2023).

4 Приложения в Google Play – Строительный калькулятор. – Текст : электронный // Google Play : [сайт]. – 2023. – URL: https://play.google.com/store/apps/details?id=calculate.willmaze.ru.build_calculate (дата обращения: 11.02.2023).

5 Приложения в Google Play – Строительный калькулятор. – Текст : электронный // Google Play : [сайт]. – 2023. – URL: https://play.google.com/store/apps/details?id=com.codart.building_calculator (дата обращения: 11.02.2023).

6 iOS или Android: что выбрать? Разработка на vc.ru. – Текст : электронный // VC : [сайт]. – 2023. – URL: https://vc.ru/dev/181153-ios-ili-android-chto-vybrat (дата обращения: 15.02.2023).

7 Operating System Market Share Worldwide. – Текст : электронный // statcounter GlobalStats : [сайт]. – 2023. – URL: https://gs.statcounter.com/os-market-share (дата обращения: 17.02.2023).

8 Kotlin Programming Language. – Текст : электронный // Kotlinlang : [сайт]. – 2023. – URL: https://kotlinlang.org/ (дата обращения 17.02.2023).

9 Как выбрать язык программирования для создания Андроид – приложения. – Текст : электронный // Хабр : [сайт]. – 2023. – URL: https://habr.com/ru/post/477578/ (дата обращения: 19.02.2023).

10 Download Android Studio & App Tools - Android Developers. – Текст : электронный // Android : [сайт]. – 2023. – URL: https://developer.android.com/studio (дата обращения 19.02.2023).

11 FAQ – OpenGL Wiki. – Текст : электронный // Khronos : [сайт]. – 2023. – URL: https://www.khronos.org/opengl/wiki/FAQ#What_is_OpenGL?/ (дата обращения: 20.04.2023).

12 Лыгина, Н.И. Информатика : учебное пособие / Н.И. Лыгина, О.В. Лауферман. – Новосибирск : НГТУ, 2017. – 84 с. – ISBN 978-5-7782-3214-3. – Текст : электронный // Лань : электронно-библиотечная система. – URL: https://e.lanbook.com/book/118216 (дата обращения: 28.05.2023). – Режим доступа: для авторизованных пользователей.

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ А

Исходный код класса активности редактора трёхмерной модели помещения

```
package com.example.ezdecor
import android.app.*
import android.content.Intent
import android.content.pm.PackageManager
import android.graphics.*
import android.opengl.GLSurfaceView
import android.os.*
import android.widget.*
import androidx.activity.result.contract.ActivityResultContracts
import androidx.appcompat.app.AppCompatActivity
import androidx.core.app.ActivityCompat
import androidx.core.content.*
import java.math.RoundingMode
import java.text.*
import java.util.*
import android.Manifest
class Editor3Activity : AppCompatActivity() {
    private val readMediaImagePermissionRequestCode = 1001
    private val cameraPermissionRequestCode = 1002
    private lateinit var renderer: MyRenderer
    private lateinit var room3D: Room3D
    private lateinit var glSurfaceView: GLSurfaceView
    private lateinit var prevButton: Button
    private lateinit var addWPButton: Button
    private lateinit var addDWButton: Button
    private lateinit var calcMaterialButton: Button
    private val doorWindowList: ArrayList<DoorWindow> = ArrayList()
    private var triangleForm = false
    private var sideC: Float = 0f
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_editor3)
        val sideA = this.intent.getFloatExtra("SideA", 0f)
        val sideB = this.intent.getFloatExtra("SideB", 0f)
        val heightRoom = this.intent.getFloatExtra("HeightR", 0f)
        triangleForm = this.intent.getBooleanExtra("triangleForm", false)
        if(triangleForm) {
            sideC = this.intent.getFloatExtra("SideC", 0f)
        }
        room3D = Room3D(sideA, sideB, sideC, heightRoom, triangleForm)
        if (!supportES2()) {
            Toast.makeText(this, "OpenGl ES 2.0 не поддерживается!",
Toast.LENGTH_LONG)
                .show()
            finish()
            return
        }
        renderer = MyRenderer(this, room3D)
        glSurfaceView = findViewById(R.id.glSurfaceView)
        glSurfaceView.setEGLContextClientVersion(2)
        glSurfaceView.setRenderer(renderer)
        prevButton = findViewById(R.id.prev_button)
        addWPButton = findViewById(R.id.addWP_button)
        addDWButton = findViewById(R.id.addDW_button)
        calcMaterialButton = findViewById(R.id.calcMaterial_button)
```

```
        prevButton.setOnClickListener {
            val intent = Intent(this, CreateActivity::class.java)
            startActivity(intent)
        }
        addWPButton.setOnClickListener {
            showImageSourceDialog()
        }
        addDWButton.setOnClickListener {
            showAddWindowDoorDialog()
        }
        calcMaterialButton.setOnClickListener {
            calculateMaterialCost()
        }
    }

    private fun showImageSourceDialog() {
        val options = arrayOf("По умолчанию", "Выбрать из галереи",
"Сфотографировать")
        AlertDialog.Builder(this)
            .setTitle("Выберите источник изображения")
            .setItems(options) { _, which ->
                when (which) {
                    0 -> setDefaultWP()
                    1 -> chooseImageFromGallery()
                    2 -> captureImage()
                }
            }
            .show()
    }
    private fun showAddWindowDoorDialog() {
        val dialogLayout = layoutInflater.inflate(R.layout.add_window_door_dialog,
null)
        val dialogBuilder = AlertDialog.Builder(this)
            .setTitle("Добавить окно или дверь")
            .setView(dialogLayout)
            .setPositiveButton("Добавить") { dialog, _ ->
                val typeSpinner =
dialogLayout.findViewById<Spinner>(R.id.typeSpinner)
                val lengthEditText =
dialogLayout.findViewById<EditText>(R.id.lengthEditText)
                val widthEditText =
dialogLayout.findViewById<EditText>(R.id.widthEditText)
                val type = typeSpinner.selectedItem.toString()
                val length = lengthEditText.text.toString().toFloatOrNull()
                val width = widthEditText.text.toString().toFloatOrNull()
                if (length != null && width != null) {
                    saveWindowDoorData(type, length, width)
                    dialog.dismiss()
                } else {
                    Toast.makeText(this, "Пожалуйста, заполните все поля",
                        Toast.LENGTH_SHORT).show()
                }
            }
            .setNegativeButton("Отмена") { dialog, _ ->
                dialog.dismiss()
            }
        val dialog = dialogBuilder.create()
        dialog.show()
    }
    private fun saveWindowDoorData(type: String, length: Float, width: Float) {
        val doorWindow = DoorWindow(type, length, width)
        doorWindowList.add(doorWindow)
```

```
        room3D.setAreaWalls(length, width)
        Toast.makeText(
            this,
            "Добавлен элемент: Тип: ${doorWindow.getType()}, " +
                    "Длина: ${doorWindow.getLength()}," +
                    " Ширина: ${doorWindow.getWidth()}",
            Toast.LENGTH_SHORT
        ).show()
    }
    private fun calculateMaterialCost() {
        val dialogLayout = layoutInflater.inflate(R.layout.material_cost_dialog,
null)
        val dialogBuilder = AlertDialog.Builder(this)
            .setView(dialogLayout)
            .setTitle("Расчет стоимости материала")
        val priceEditText =
dialogLayout.findViewById<EditText>(R.id.priceEditText)
        val totalCostTextView =
dialogLayout.findViewById<TextView>(R.id.totalCostTextView)
        val calculateButton =
dialogLayout.findViewById<Button>(R.id.calculateButton)
        val cancelButton = dialogLayout.findViewById<Button>(R.id.cancelButton)
        val totalAreaTextView =
dialogLayout.findViewById<TextView>(R.id.totalAreaTextView)
        val countOfRolls =
dialogLayout.findViewById<TextView>(R.id.count_of_rolls)
        val lengthEditText =
dialogLayout.findViewById<TextView>(R.id.length_of_wp_edit)
        val widthEditText =
dialogLayout.findViewById<TextView>(R.id.width_of_wp_edit)
        val dialog = dialogBuilder.create()
        calculateButton.setOnClickListener {
            val price = priceEditText.text.toString().toFloatOrNull()
            val length = lengthEditText.text.toString().toFloatOrNull()
            val width = widthEditText.text.toString().toFloatOrNull()
            if ((price != null) && (length != null) && (width != null)) {
                val countRolls = room3D.getAreaWalls()/(length*width)
                val df = DecimalFormat("#")
                df.roundingMode = RoundingMode.UP
                val countRollsInt = df.format(countRolls).toInt()
                val totalCost = price * countRollsInt
                totalAreaTextView.text = getString(R.string.total_square_e3a,
room3D.getAreaWalls())
                countOfRolls.text =
getString(R.string.count_of_rolls_e3a,countRollsInt)
                totalCostTextView.text = getString(R.string.total_cost_e3a,
totalCost)
            } else {
                totalCostTextView.text = "Введите корректную цену, длину или
ширину"
            }
        }
        cancelButton.setOnClickListener{
            dialog.dismiss()
        }
        dialog.show()
    }
    private fun supportES2(): Boolean {
        val activityManager = getSystemService(ACTIVITY_SERVICE) as
ActivityManager
        val configurationInfo = activityManager.deviceConfigurationInfo
        return configurationInfo.reqGlEsVersion >= 0x20000
```

```
    }
    override fun onPause() {
        super.onPause()
        glSurfaceView.onPause()
    }
    override fun onResume() {
        super.onResume()
        glSurfaceView.onResume()
    }
    private fun setDefaultWP() {
        if (renderer.getResourceID() == R.drawable.wallpapers) {
            return
        }else{
            glSurfaceView.queueEvent {
                renderer.updateWallTexture(R.drawable.wallpapers)
            }
            glSurfaceView.requestRender()
        }
    }
    private val galleryLauncher =
registerForActivityResult(ActivityResultContracts.GetContent()) { imageUri ->
        imageUri?.let {
            val inputStream = contentResolver.openInputStream(imageUri)
            val options = BitmapFactory.Options()
            options.inScaled = false
            val bitmap = BitmapFactory.decodeStream(inputStream, null, options)

            glSurfaceView.queueEvent {
                renderer.updateWallTextureBitmap(bitmap!!)
            }
            glSurfaceView.requestRender()

            inputStream?.close()
        }
    }
    private fun chooseImageFromGallery() {
        if ((ContextCompat.checkSelfPermission(this,
Manifest.permission.READ_MEDIA_IMAGES)
                    != PackageManager.PERMISSION_GRANTED)
            || (ContextCompat.checkSelfPermission(this,
Manifest.permission.READ_EXTERNAL_STORAGE)
                    != PackageManager.PERMISSION_GRANTED)){
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU) {
                ActivityCompat.requestPermissions(this,
arrayOf(Manifest.permission.READ_MEDIA_IMAGES),
readMediaImagePermissionRequestCode)
            }
            else{
                ActivityCompat.requestPermissions(this,
arrayOf(Manifest.permission.READ_EXTERNAL_STORAGE),
readMediaImagePermissionRequestCode)
            }
            galleryLauncher.launch("image/*")
        } else {
            galleryLauncher.launch("image/*")
        }
    }
    private val cameraLauncher =
registerForActivityResult(ActivityResultContracts.TakePicturePreview()) { result -
>
            if (result != null){
                glSurfaceView.queueEvent {
```

```
                    renderer.updateWallTextureBitmap(result)
                }
                glSurfaceView.requestRender()
            }
            else{
                return@registerForActivityResult
            }
    }
    private fun captureImage() {
        if (ContextCompat.checkSelfPermission(this, Manifest.permission.CAMERA) !=
PackageManager.PERMISSION_GRANTED) {
            ActivityCompat.requestPermissions(this,
arrayOf(Manifest.permission.CAMERA), cameraPermissionRequestCode)
            cameraLauncher.launch(null)
        } else {
            cameraLauncher.launch(null)
        }
    }
}
```

ПРИЛОЖЕНИЕ Б

Исходный код класса средства визуализации

```
package com.example.ezdecor
import android.content.Context
import android.graphics.Bitmap
import android.opengl.*
import android.opengl.Matrix
import android.os.SystemClock
import java.nio.*
import javax.microedition.khronos.egl.EGLConfig
import javax.microedition.khronos.opengles.GL10
import kotlin.math.*
class MyRenderer(private val context: Context, private val room3D: Room3D) :
GLSurfaceView.Renderer {
    private var newWP: Boolean = false
    private var resourceID: Int = R.drawable.wallpapers
    private lateinit var bitmapFile: Bitmap
    private var lengthA: Float = room3D.getSideA()*0.1f
    private var lengthB: Float = room3D.getSideB()*0.1f
    private var height: Float = room3D.getHeightRoom()*0.1f
    private val positionCount = 3
    private val textureCount = 2
    private val stride = (positionCount + textureCount) * 4
    private var vertexData: FloatBuffer? = null
    private var aPositionLocation = 0
    private var aTextureLocation = 0
    private var uTextureUnitLocation = 0
    private var uTextureUnitLocationBottom = 0
    private var uMatrixLocation = 0
    private var programId = 0
    private val mProjectionMatrix = FloatArray(16)
    private val mViewMatrix = FloatArray(16)
    private val mMatrix = FloatArray(16)
    private var texture = 0
    private var texture2 = 0
    private var texture3 = 0
    private var aTextureLocationBottom = 0
    private val time: Long = 10000
    fun getResourceID(): Any {
        return resourceID
    }
    override fun onSurfaceCreated(arg0: GL10, arg1: EGLConfig) {
        GLES20.glClearColor(0f, 0f, 0f, 1f)
        GLES20.glEnable(GLES20.GL_DEPTH_TEST)
        createAndUseProgram()
        locations()
        prepareData(newWP)
        bindData()
        createViewMatrix()
    }
    override fun onSurfaceChanged(arg0: GL10, width: Int, height: Int) {
        GLES20.glViewport(0, 0, width, height)
        createProjectionMatrix(width, height)
        bindMatrix()
    }
    private fun prepareData(bitmap: Boolean) {
        val vertices: FloatArray
        if (room3D.getTriangleForm()){
            val heightTriangle: Float = room3D.getHeightTriangle()*0.1f
            val rightPartA: Float = room3D.getRightPartA()*2*0.1f
            vertices = floatArrayOf( // Координаты вершин и текстурных координат
боковых граней
```

```
                -lengthA, height, heightTriangle, 0f, 0f, //передний квадрат стены
                -lengthA, -height, heightTriangle, 0f, 1f,
                lengthA, height, heightTriangle, 1f, 0f,
                lengthA, -height, heightTriangle, 1f, 1f,
                lengthA, height, heightTriangle, 0f, 0f, //правый квадрат стены
                lengthA, -height, heightTriangle, 0f, 1f,
                lengthA-rightPartA, height, -heightTriangle, 1f, 0f,
                lengthA-rightPartA, -height, -heightTriangle, 1f, 1f,
                lengthA-rightPartA, height, -heightTriangle, 0f, 0f, //левый
квадрат стены
                lengthA-rightPartA, -height, -heightTriangle, 0f, 1f,
                -lengthA, height, heightTriangle, 1f, 0f,
                -lengthA, -height, heightTriangle, 1f, 1f,
                -lengthA*0.99f, height*0.99f, heightTriangle*0.99f, 0f, 0f,
//передний квадрат
                -lengthA*0.99f, -height*0.99f, heightTriangle*0.99f, 0f, 1f,
                lengthA*0.99f, height*0.99f, heightTriangle*0.99f, 1f, 0f,
                lengthA*0.99f, -height*0.99f, heightTriangle*0.99f, 1f, 1f,
                lengthA*0.99f, height*0.99f, heightTriangle*0.99f, 0f, 0f,
//правый квадрат
                lengthA*0.99f, -height*0.99f, heightTriangle*0.99f, 0f, 1f,
                (lengthA-rightPartA)*0.99f, height*0.99f, -heightTriangle*0.99f,
1f, 0f,
                (lengthA-rightPartA)*0.99f, -height*0.99f, -heightTriangle*0.99f,
1f, 1f,
                (lengthA-rightPartA)*0.99f, height*0.99f, -heightTriangle*0.99f,
0f, 0f, //левый квадрат
                (lengthA-rightPartA)*0.99f, -height*0.99f, -heightTriangle*0.99f,
0f, 1f,
                -lengthA*0.99f, height*0.99f, heightTriangle*0.99f, 1f, 0f,
                -lengthA*0.99f, -height*0.99f, heightTriangle*0.99f, 1f, 1f,
                // Координаты вершин и текстурных координат нижней грани
                -lengthA, -height, heightTriangle, 0f, 1f,
                lengthA, -height, heightTriangle, 1f, 0f,
                lengthA-rightPartA, -height, -heightTriangle, 0f, 0f)
        }
        else{
            vertices = floatArrayOf( // Координаты вершин и текстурных координат
боковых граней
                -lengthA, height, lengthB, 0f, 0f, //передний квадрат стены
                -lengthA, -height, lengthB, 0f, 1f,
                lengthA, height, lengthB, 1f, 0f,
                lengthA, -height, lengthB, 1f, 1f,
                lengthA, height, lengthB, 0f, 0f, //правый квадрат стены
                lengthA, -height, lengthB, 0f, 1f,
                lengthA, height, -lengthB, 1f, 0f,
                lengthA, -height, -lengthB, 1f, 1f,
                -lengthA, height, -lengthB, 0f, 0f, //задний квадрат стены
                -lengthA, -height, -lengthB, 0f, 1f,
                lengthA, height, -lengthB, 1f, 0f,
                lengthA, -height, -lengthB, 1f, 1f,
                -lengthA, height, -lengthB, 0f, 0f, //левый квадрат стены
                -lengthA, -height, -lengthB, 0f, 1f,
                -lengthA, height, lengthB, 1f, 0f,
                -lengthA, -height, lengthB, 1f, 1f,
                -lengthA*0.99f, height*0.99f, lengthB*0.99f, 0f, 0f, //передний
квадрат
                -lengthA*0.99f, -height*0.99f, lengthB*0.99f, 0f, 1f,
                lengthA*0.99f, height*0.99f, lengthB*0.99f, 1f, 0f,
                lengthA*0.99f, -height*0.99f, lengthB*0.99f, 1f, 1f,
                lengthA*0.99f, height*0.99f, lengthB*0.99f, 0f, 0f, //правый
квадрат
```

```
                lengthA*0.99f, -height*0.99f, lengthB*0.99f, 0f, 1f,
                lengthA*0.99f, height*0.99f, -lengthB*0.99f, 1f, 0f,
                lengthA*0.99f, -height*0.99f, -lengthB*0.99f, 1f, 1f,
                -lengthA*0.99f, height*0.99f, -lengthB*0.99f, 0f, 0f, //задний
квадрат
                -lengthA*0.99f, -height*0.99f, -lengthB*0.99f, 0f, 1f,
                lengthA*0.99f, height*0.99f, -lengthB*0.99f, 1f, 0f,
                lengthA*0.99f, -height*0.99f, -lengthB*0.99f, 1f, 1f,
                -lengthA*0.99f, height*0.99f, -lengthB*0.99f, 0f, 0f, //левый
квадрат
                -lengthA*0.99f, -height*0.99f, -lengthB*0.99f, 0f, 1f,
                -lengthA*0.99f, height*0.99f, lengthB*0.99f, 1f, 0f,
                -lengthA*0.99f, -height*0.99f, lengthB*0.99f, 1f, 1f,
                // Координаты вершин и текстурных координат нижней грани
                -lengthA, -height, lengthB, 0f, 1f,
                lengthA, -height, lengthB, 1f, 1f,
                -lengthA, -height, -lengthB, 0f, 0f,
                lengthA, -height, -lengthB, 1f, 0f)
        }
        vertexData = ByteBuffer.allocateDirect(vertices.size * 4)
            .order(ByteOrder.nativeOrder())
            .asFloatBuffer()
        vertexData?.put(vertices)
        texture = TextureUtils.loadTexture(context, R.drawable.wall)
        texture2 = if (bitmap){
            TextureUtils.loadTexture(bitmapFile)
        } else{
            TextureUtils.loadTexture(context, resourceID)
        }
        texture3 = TextureUtils.loadTexture(context, R.drawable.floor)
    }
    private fun createAndUseProgram() {
        val vertexShaderId: Int =
            ShaderUtils.createShader(context, GLES20.GL_VERTEX_SHADER,
R.raw.vertex_shader)
        val fragmentShaderId: Int =
            ShaderUtils.createShader(context, GLES20.GL_FRAGMENT_SHADER,
R.raw.fragment_shader)
        programId = ShaderUtils.createProgram(vertexShaderId, fragmentShaderId)
        GLES20.glUseProgram(programId)
    }
    private fun locations() {
        aPositionLocation = GLES20.glGetAttribLocation(programId, "a_Position")
        aTextureLocation = GLES20.glGetAttribLocation(programId, "a_Texture")
        aTextureLocationBottom = GLES20.glGetAttribLocation(programId,
"a_TextureBottom")
        uTextureUnitLocation = GLES20.glGetUniformLocation(programId,
"u_TextureUnit")
        uTextureUnitLocationBottom = GLES20.glGetUniformLocation(programId,
"u_TextureUnitBottom")
        uMatrixLocation = GLES20.glGetUniformLocation(programId, "u_Matrix")
    }
    private fun bindData() {
        vertexData?.position(0)
        GLES20.glVertexAttribPointer(aPositionLocation, positionCount,
GLES20.GL_FLOAT, false, stride, vertexData)
        GLES20.glEnableVertexAttribArray(aPositionLocation)
        vertexData?.position(positionCount)
        GLES20.glVertexAttribPointer(aTextureLocation, textureCount,
GLES20.GL_FLOAT, false, stride, vertexData)
        GLES20.glEnableVertexAttribArray(aTextureLocation)
    }
```

```
    private fun createProjectionMatrix(width: Int, height: Int) {
        val ratio: Float
        var left = -1f
        var right = 1f
        var bottom = -1f
        var top = 1f
        val near = 2f
        val far = 12f
        if (width > height) {
            ratio = width.toFloat() / height
            left *= ratio
            right *= ratio
        } else {
            ratio = height.toFloat() / width
            bottom *= ratio
            top *= ratio
        }
        Matrix.frustumM(mProjectionMatrix, 0, left, right, bottom, top, near, far)
    }
    private fun createViewMatrix() {
        val timeF: Float = (SystemClock.uptimeMillis() % time).toFloat() / time
        val angle = timeF * 2 * PI.toFloat()
        // точка положения камеры
        val eyeX = cos(angle) * 1.3f
        val eyeY = 1.2f
        var eyeZ = sin(angle) * 2.5f
        eyeZ *= if (lengthA>height) lengthA else height
        // точка направления камеры
        val centerX = 0f
        val centerY = 0f
        val centerZ = 0f
        // up-вектор
        val upX = 0f
        val upY = 1f
        val upZ = 0f
        Matrix.setLookAtM(mViewMatrix, 0, eyeX, eyeY, eyeZ, centerX, centerY,
centerZ,
            upX, upY, upZ)
    }
    private fun bindMatrix() {
        Matrix.multiplyMM(mMatrix, 0, mProjectionMatrix, 0, mViewMatrix, 0)
        GLES20.glUniformMatrix4fv(uMatrixLocation, 1, false, mMatrix, 0)
    }
    override fun onDrawFrame(arg0: GL10) {
        createViewMatrix()
        bindMatrix()
        GLES20.glClear(GLES20.GL_COLOR_BUFFER_BIT or GLES20.GL_DEPTH_BUFFER_BIT)
        GLES20.glActiveTexture(GLES20.GL_TEXTURE0)
        GLES20.glBindTexture(GLES20.GL_TEXTURE_2D, texture)
        if (room3D.getTriangleForm()){
            GLES20.glDrawArrays(GLES20.GL_TRIANGLE_STRIP, 0, 4)
            GLES20.glDrawArrays(GLES20.GL_TRIANGLE_STRIP, 4, 4)
            GLES20.glDrawArrays(GLES20.GL_TRIANGLE_STRIP, 8, 4)
        }else {
            GLES20.glDrawArrays(GLES20.GL_TRIANGLE_STRIP, 0, 4)
            GLES20.glDrawArrays(GLES20.GL_TRIANGLE_STRIP, 4, 4)
            GLES20.glDrawArrays(GLES20.GL_TRIANGLE_STRIP, 8, 4)
            GLES20.glDrawArrays(GLES20.GL_TRIANGLE_STRIP, 12, 4)
        }
        GLES20.glBindTexture(GLES20.GL_TEXTURE_2D, texture2)
        if (room3D.getTriangleForm()){
            GLES20.glDrawArrays(GLES20.GL_TRIANGLE_STRIP, 12, 4)
```

```
                GLES20.glDrawArrays(GLES20.GL_TRIANGLE_STRIP, 16, 4)
                GLES20.glDrawArrays(GLES20.GL_TRIANGLE_STRIP, 20, 4)
            }else {
                GLES20.glDrawArrays(GLES20.GL_TRIANGLE_STRIP, 16, 4)
                GLES20.glDrawArrays(GLES20.GL_TRIANGLE_STRIP, 20, 4)
                GLES20.glDrawArrays(GLES20.GL_TRIANGLE_STRIP, 24, 4)
                GLES20.glDrawArrays(GLES20.GL_TRIANGLE_STRIP, 28, 4)
            }
            GLES20.glBindTexture(GLES20.GL_TEXTURE_2D, texture3)
            if (room3D.getTriangleForm()){
                GLES20.glDrawArrays(GLES20.GL_TRIANGLE_STRIP, 24, 3)
            }else {
                GLES20.glDrawArrays(GLES20.GL_TRIANGLE_STRIP, 32, 4)
                GLES20.glDrawArrays(GLES20.GL_TRIANGLE_STRIP, 36, 4)
            }
        }
        fun updateWallTexture(textureId: Int) {
            resourceID = textureId
            newWP = false
        }
        fun updateWallTextureBitmap(bitmap: Bitmap) {
            bitmapFile = bitmap
            newWP = true
        }
    }
```

ПРИЛОЖЕНИЕ В

Исходный код класса активности создания трёхмерной модели помещения

```
package com.example.ezdecor
import android.app.AlertDialog
import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.View
import android.widget.*
class CreateActivity : AppCompatActivity() {
    private lateinit var prevButton: Button
    private lateinit var rectangleButton: Button
    private lateinit var triangleButton: Button
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_create)
        prevButton = findViewById(R.id.prev_button)
        rectangleButton = findViewById(R.id.rectangle_button)
        triangleButton = findViewById(R.id.triangle_button)
        prevButton.setOnClickListener {
            val intent = Intent(this, MainActivity::class.java)
            startActivity(intent)
        }
        rectangleButton.setOnClickListener {
            createDialogInputRectangle()
        }
        triangleButton.setOnClickListener {
            createDialogInputTriangle()
        }
    }
    private fun createDialogInputTriangle(){
        val dialogInput = AlertDialog.Builder(this)
        val inflater = layoutInflater
        dialogInput.setTitle(R.string.input_param_title)
        val dialogLayout = inflater.inflate(R.layout.input_param, null)
        val sideA  = dialogLayout.findViewById<EditText>(R.id.sideA)
        val sideB  = dialogLayout.findViewById<EditText>(R.id.sideB)
        val sideC  = dialogLayout.findViewById<EditText>(R.id.sideC)
        val heightRoom  = dialogLayout.findViewById<EditText>(R.id.heightRoom)
        dialogInput.setView(dialogLayout)
        dialogInput.setNegativeButton(R.string.prev_button){ dialog, _ ->
            dialog.dismiss()
        }
        dialogInput.setPositiveButton(R.string.next_button) { _, _ ->
            val a = sideA.text.toString().toFloatOrNull()
            val b = sideB.text.toString().toFloatOrNull()
            val c = sideC.text.toString().toFloatOrNull()
            val h = heightRoom.text.toString().toFloatOrNull()
            if (a != null && b != null && c != null && h != null && a > 0f && b >
0f && c > 0f && h > 0f && isTriangleValid(a, b, c)) {
                val intent = Intent(this, Editor3Activity::class.java)
                intent.putExtra("SideA", a)
                intent.putExtra("SideB", b)
                intent.putExtra("SideC", c)
                intent.putExtra("HeightR", h)
                intent.putExtra("triangleForm", true)
                startActivity(intent)
            } else {
                Toast.makeText(this, "Такая фигура существует",
 Toast.LENGTH_SHORT).show()
            }
        }
```

```
        dialogInput.show()
    }
    private fun isTriangleValid(a: Float, b: Float, c: Float): Boolean {
        return (a + b > c) && (b + c > a) && (c + a > b)
    }
    private fun createDialogInputRectangle(){
        val dialogInput = AlertDialog.Builder(this)
        val inflater = layoutInflater
        dialogInput.setTitle(R.string.input_param_title)
        val dialogLayout = inflater.inflate(R.layout.input_param, null)
        val sideA  = dialogLayout.findViewById<EditText>(R.id.sideA)
        val sideB  = dialogLayout.findViewById<EditText>(R.id.sideB)
        val heightRoom  = dialogLayout.findViewById<EditText>(R.id.heightRoom)
        val sideCText = dialogLayout.findViewById<TextView>(R.id.sideCText)
        val sideC = dialogLayout.findViewById<EditText>(R.id.sideC)
        sideCText.visibility = View.GONE
        sideC.visibility = View.GONE
        dialogInput.setView(dialogLayout)
        dialogInput.setNegativeButton(R.string.prev_button){ dialog, _ ->
            dialog.dismiss()
        }
        dialogInput.setPositiveButton(R.string.next_button){ _, _ ->
            val a = sideA.text.toString().toFloatOrNull()
            val b = sideB.text.toString().toFloatOrNull()
            val h = heightRoom.text.toString().toFloatOrNull()
            if (a != null && b != null && h != null && a > 0f && b > 0f && h > 0f
) {
                val intent = Intent(this, Editor3Activity::class.java)
                intent.putExtra("SideA", a)
                intent.putExtra("SideB", b)
                intent.putExtra("HeightR", h)
                intent.putExtra("triangleForm", false)
                startActivity(intent)
            } else {
                Toast.makeText(this, "Такая фигура существует",
Toast.LENGTH_SHORT).show()
            }
        }
        dialogInput.show()
    }
}
```