

**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**  
**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ**  
**УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**  
«Национальный исследовательский ядерный университет «МИФИ»  
**Обнинский институт атомной энергетики –**  
филиал федерального государственного автономного образовательного учреждения высшего  
образования «Национальный исследовательский ядерный университет «МИФИ»  
(ИАТЭ НИЯУ МИФИ)

Отделение Интеллектуальные кибернетические системы  
Направление подготовки Информационные системы и технологии

**Выпускная квалификационная работа —  
бакалаврская работа**

по направлению подготовки **09.03.02 Информационные  
системы и технологии**

Направленность (профиль) **Информационные технологии**

**«Разработка мобильного приложения для  
преобразования 2D фотографий в 3D вид»**

Выполнил:

студент гр. ИС-Б14 \_\_\_\_\_ Кузнецов А.В.

Руководитель ВКР,  
доцент отделения ИКС, к.т.н.

\_\_\_\_\_ Мирзеабасов О.А.

Нормоконтроль

\_\_\_\_\_ Бязрова В.Ф.

Выпускная квалификационная  
работа допущена к защите

\_\_\_\_\_

Руководитель  
образовательной программы  
09.03.02 Информационные системы  
и технологии  
доцент отделения ИКС, к.т.н.

\_\_\_\_\_ Мирзеабасов О.А.

Обнинск, 2018 г

# **РЕФЕРАТ**

50 стр., 0 табл., 20 рис., 12 ист.

ANDROID, JAVA, МОБИЛЬНОЕ ПРИЛОЖЕНИЕ, ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС, UI/UX, КАРТА ГЛУБИНЫ, АНАЛИЗ, 3D, АНИМАЦИЯ, GIF

Настоящая работа посвящена изучению методов получения трехмерных изображений из двумерных и разработке пользовательского интерфейса мобильного приложения для преобразования 2D фотографий в 3D вид.

Разработанная программа дает возможность преобразовывать простую фотографию в объемную картинку, в виде GIF.

## **ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ**

ВКР — Выпускная квалификационная работа

UI (user interface) — Пользовательский интерфейс

UX (user experience) — Пользовательский опыт

ПО — Программное обеспечение

ОС — Операционная система

GIF — Graphics Interchange Format

# СОДЕРЖАНИЕ

<b>ВВЕДЕНИЕ</b>	<b>6</b>
<b>1. Методы получения трехмерных изображений из двумерных</b>	<b>8</b>
1.1. Восстановление глубины из одного расфокусированного изображения . . . . .	11
1.2. Модель дефокусировки . . . . .	12
1.3. Оценка размытости . . . . .	13
1.4. Интерполяция глубины . . . . .	14
1.5. Тестирование стабильности и качества алгоритма . . . . .	15
1.6. Оценка достоверности построения карты глубины . . . . .	16
<b>2. Описание предметной области</b>	<b>18</b>
2.1. Android Studio . . . . .	18
2.2. Java . . . . .	19
2.3. Технические требования. . . . .	19
2.4. Изучение основ разработки мобильного приложения в среде Android Studio . . . . .	20
2.5. Основные компоненты пользовательского интерфейса мобильного приложения в Android . . . . .	22
<b>3. Анализ существующих продуктов со схожим функционалом</b>	<b>24</b>
3.1. Make It 3d Free . . . . .	24
3.2. 3D Effect . . . . .	25
3.3. Motion Photo . . . . .	26
3.4. Fyuse . . . . .	27
3.5. Вывод . . . . .	27
<b>4. Разработка пользовательского интерфейса мобильного приложения (UI/UX)</b>	<b>28</b>
4.1. Анализ интерфейсов смежных приложений . . . . .	28
4.1.1. Prisma . . . . .	28
4.1.2. Vinci . . . . .	30
4.1.3. Pixlr . . . . .	31

4.1.4. Snapster . . . . .	32
4.2. Разработка пользовательского интерфейса мобильного приложения (UI/UX) . . . . .	33
4.3. Проектирование структуры приложения . . . . .	36
<b>5. Разработка мобильного приложения</b>	<b>39</b>
5.1. Разработка концепции и архитектуры мобильного приложения, предназначенных для преобразования 2D в 3D . . . . .	39
5.2. Разработка мобильного приложения . . . . .	43
<b>ЗАКЛЮЧЕНИЕ</b>	<b>45</b>
<b>СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ</b>	<b>46</b>
<b>Приложение А</b>	<b>47</b>
<b>Приложение В</b>	<b>49</b>

# **ВВЕДЕНИЕ**

В настоящее время активно развиваются технологии визуализации графической информации в трехмерном виде. В дополнение к существующим устройствам и технологиям съемки макрообъектов я предлагаю создать приложение для «Android» устройств, с помощью которого пользователи смогут выкладывать в социальных сетях и делиться со знакомыми своими фотографиями в объемном виде.

Основным результатом выполнения проекта будет мобильное приложение, предназначенное для преобразования 2D изображений в 3D вид. Приложение будет распространяться с помощью его размещения в Google Play (для Android-устройств). Соответственно, в качестве основных потребителей создаваемой продукции следует рассматривать владельцев мобильных устройств, которые любят использовать свой телефон или планшет в качестве фотоаппарата. Более того, то подмножество этих пользователей, которые, помимо фотографирования, активно обрабатывают свои фото средствами мобильного устройства и активно делятся этими результатами с друзьями посредством социальных сетей.

Поэтому главной целью текущей ВКР является исследование преобразования двумерных изображений в трехмерные и разработке современного, функционального и удобного пользовательского интерфейса.

В связи с этим, передо мной были поставлены следующие задачи:

- 1) Изучение работы алгоритма определения глубины изображения
- 2) Тестирование стабильности и качества алгоритма преобразования 2D фотографий в 3D
- 3) Программная реализация алгоритма преобразования 2D фотографии в 3D
- 4) Приведение библиотеки в приемлемый вид для использования во «внешнем» приложении.
- 5) Анализ существующих продуктов со схожим функционалом
- 6) Разработка пользовательского интерфейса мобильного приложения

- 7) Разработка концепции и архитектуры мобильного приложения, предна-  
значенного для преобразования 2D в 3D
- 8) Проектирование структуры приложения
- 9) Изучение основ разработки мобильного приложения в среде Android  
Studio
- 10) Создание мобильного приложения

# 1. Методы получения трехмерных изображений из двумерных

В ходе выполнения проекта, главным образом, решается задача преобразования двумерных изображений в трехмерные на мобильных устройствах.

В последние годы заметное место в области преобразования и фильтрации изображений занимает задача преобразования двумерных изображений в трехмерные. На сегодняшний день в мире для этого разработаны различные методики, которые позволяют автоматически создавать так называемые «карты глубины» [2] (рисунок 1) для двумерных изображений, основываясь на свойствах этого изображение и на некоторых предположениях о характере сцены.



Рисунок 1 – карта глубины

В частности, были проведены исследования следующих методов получения трехмерных изображений из двумерных:

- Предположение о том, что изображение имеет линейную перспективу;
- Предположение о том, что снимок сделан на открытом пространстве и использование модели рассеяния световых лучей в атмосфере;
- Обнаружение теней и восстановление по ним карты глубины;
- Обнаружение перекрытий объектов на изображении и используя эту информацию восстановлении карты глубины;

- Использование моделей пространственных искажений заданных текстур;
- Использование билатеральных симметричных шаблонов;
- Использование статистических методов для обучения текстурных шаблонов, на различных расстояниях от объектива и другие методы.

Практически все эти методы характеризуются достаточно узким характером сцен, которые могут быть реализованы для преобразования в трехмерное изображение.

Проведенные предварительные исследования показали, что одним из наиболее перспективных методов преобразования изображений в трехмерные считается метод «дефокусировки», который предполагает, что близкие объекты находятся в фокусе, а более удаленные объекты имеют большее размытие. Используя информацию о размытии той или иной точки на изображении можно предположить о том, насколько далеко она находится от объектива. Используя метод дефокусировки можно построить карту глубины для любой макрофотографии.

Известные методы получения карты глубины, использующие дефокус удаленных от объектива объектов, основаны на следующей цепочке преобразования изображения [3]:

- 1) Обнаружение краев объектов с использованием фильтра Канни.
- 2) Для каждой точки найденного края выполняется оценка расстояния, до нее с использованием гауссовского размытия. Таким образом строится так называемая разреженная карта глубины, которая несет информацию о расстоянии до объектива в некоторых точках изображения.
- 3) Разреженная карта глубины с использованием интерполяции превращается в так называемую «плотную карту глубины», которая уже пригодна для построения трехмерного изображения сцены с любого ракурса.

Реализация мобильного приложения позволяющего оперативно переводить стандартные фотоизображения в 3D-снимки чрезвычайно актуально и востребовано. С другой стороны, на сегодняшний день имеется определенный

научный задел по разработке алгоритмов преобразования 2D -3D. В частности известен ряд различных методик, которые позволяют автоматически создавать «карты глубины» для двумерных изображений, основываясь на свойствах этого изображения, и на некоторых предположениях о характере сцены. Например метод дефокусировки позволяет построить карту глубины фотографии. Вместе с тем информации о программно реализованных в том числе мобильных приложениях крайне мало.

Рассмотренный метод преобразования (см. пункты 1-3) предыдущего раздела страдает двумя существенными недостатками[4].

- 1) Разреженная карта глубины получается не всегда гладкой, что приводит к ошибкам последующей интерполяции и, в итоге, к ошибкам в карте глубины.
- 2) Окончательная интерполяция для построения требует больших вычислительных затрат и до последнего времени не имела возможности быть встроенной в мобильные приложения.

Основываясь на этих двух недостатках существующего метода дефокусировки изображения, требуется провести научное исследование в следующих направлениях:

- 1) Адаптивное сглаживание разреженной карты глубины. С использованием методов кластеризации краев изображения, в протяженные структуры, глубина точек которых должна подчиняться некоторому закону, а не быть случайной от точки к точке, как в существующем методе.
- 2) Найти способ снижения вычислительных затрат для выполнения двумерной интерполяции при построении плотной карты глубины.
- 3) Нам представляется, что для преобразования двумерного видеоролика в трехмерное представление необходимо разработать методику передачи разреженной карты глубины (с учетом сглаживания) от кадра к кадру.

Таким образом предлагаемый Проект на сегодняшний день актуален и содержит помимо научной алгоритмической составляющей широкие перспективы коммерциализации.

## 1.1. Восстановление глубины из одного расфокусированного изображения

Рассмотрим сложную задачу восстановления глубины из одного расфокусированного изображения. Входное расфокусированное изображение повторно размыто с использованием гауссова ядра, а значение размытости размытия может быть получено из коэффициента градиента между входными и повторно размытыми изображениями. Распространяя количество размытия в крайних положениях на все изображение, можно восстановить всю карту глубины сцены.

Результат восстановления глубины нашего метода. (рисунок 2) Большая интенсивность означает большую глубину на всех картах глубины



Рисунок 2 – Входное изображение и карта глубины

Сосредоточимся на более сложной проблеме восстановления относительной глубины из одного расфокусированного изображения, захваченного некалиброванной обычной камерой. Метод обратной диффузии [7] моделирует размытие дефокусировки в качестве процесса диффузии тепла и использует неоднородную диффузию тепла для оценки размытости размытия в краевых положениях. В отличие от этого, моделируем размытие дефокусировки как размытие 2D Gaussian. Входное изображение повторно размывается с использованием известного гауссовского размытия, и рассчитывается коэффициент градиента между входными и повторно размытыми изображениями. Величина размытия в краевых местоположениях может быть получена из отношения.

Рассмотрим эффективный метод оценки размытия, основанный на гаус-

совском градиентном соотношении, и показываем, что он устойчив к шуму, неточному расположению краев и помехам от соседних ребер. Без каких-либо изменений в камерах или при использовании дополнительного освещения наш метод позволяет получить карту глубины сцены, используя только одно расфокусированное изображение, снятое обычной камерой. Как показано (рисунок 2), этот метод может извлекать карту глубины сцены с довольно высокой степенью точности.

## 1.2. Модель дефокусировки

Оцениваем размытие размытия в местах краев и предполагаем, что края являются ступенчатыми краями. Идеальный край шага может быть смоделирован как:

$$f(x) = Au(x) + B \quad (1)$$

где  $u(x)$  - ступенчатая функция. А и В - амплитуда и смещение края соответственно. Обратите внимание, что ребро расположено в точке  $x = 0$ .

Предположим, что фокус и дефокусировка подчиняются тонкой модели объектива [8]. Когда объект размещается на расстоянии фокусировки  $d_f$ , все лучи от точки объекта будут сходиться к одной точке датчика, и изображение будет резким. Лучи из точки другого объекта на расстоянии  $d$  достигают нескольких точек датчика и приводят к размытому изображению. Размытый рисунок зависит от формы апертуры и называется кругом путаницы (CoC) [8]. Диаметр CoC характеризует величину расфокусировки и может быть записан как

$$c = \frac{|d - d_f|}{d} \frac{f_0^2}{N(d_f - f_0)} \quad (2)$$

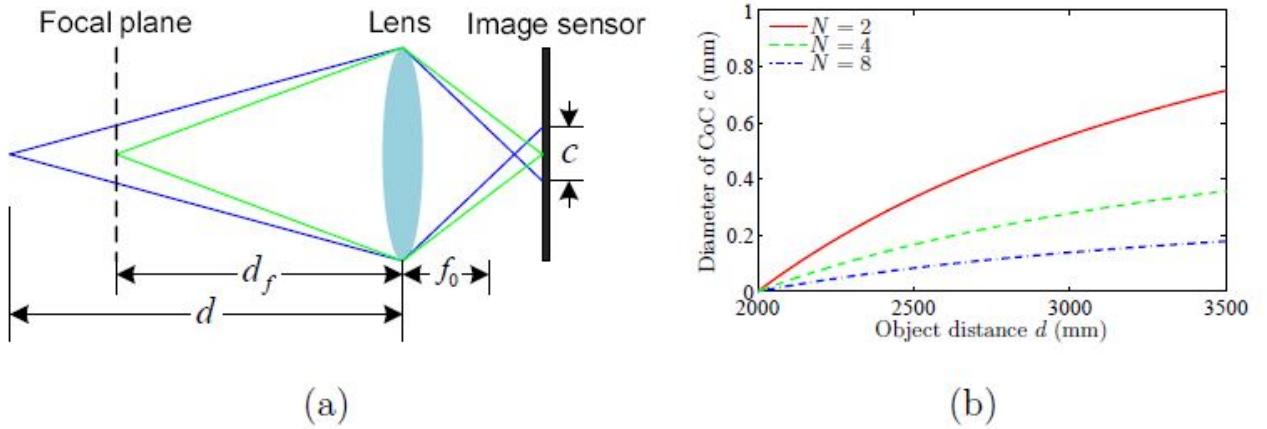


Рисунок 3 – Тонкая модель объектива

где  $f_0$  и  $N$  - фокусное расстояние и номер остановки камеры соответственно. На рисунке 3 показаны фокус и расфокусировка для модели тонких линз и как изменяется диаметр круга замешательства с  $d$  и  $N$ , при фиксированном  $f_0$  и  $d_f$ . Как мы видим, диаметр СоС с является нелинейной монотонно возрастающей функцией расстояния объекта  $d$ . Размытие дефокусировки может быть смоделировано как свертка острого изображения с функцией распределения точек (PSF). PSF обычно аппроксимируется гауссовой функцией  $g(x, \sigma)$ , где стандартное отклонение  $\sigma = k c$  пропорционально диаметру СоС  $c$ . Используем  $\sigma$  как меру глубины сцены. Следовательно, размытие ребра  $i(x)$  можно представить следующим образом,

$$i(x) = f(x) \otimes g(x, \sigma) \quad (3)$$

### 1.3. Оценка размытости

На рисунке 4 показан обзор метода оценки размытия. Граница шага повторно размыта с использованием гауссова ядра с известным стандартным отклонением. затем рассчитывается соотношение между величиной градиента края ступени и ее размытой версией. Это максимальное значение в краевом положении. Используя максимальное значение, можем вычислить величину размытости дефокусировки в местоположении края.

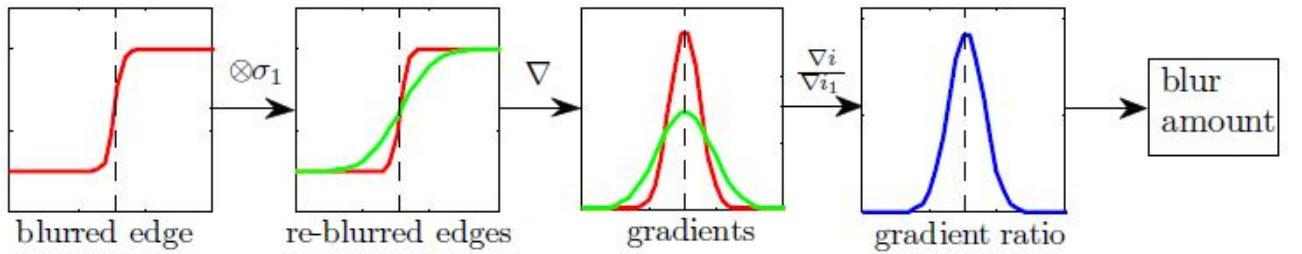


Рисунок 4 – Оценка размытости, черная штриховая линия обозначает местоположение края

Используем двумерное изотропное гауссовское ядро для повторного размытия и величину градиента можно вычислить следующим образом:

$$\|\nabla i(x, y)\| = \sqrt{\nabla i_x^2 + \nabla i_y^2} \quad (4)$$

где  $\nabla i_x$  и  $\nabla i_y$ - градиенты вдоль направлений x и y соответственно. Устанавливаем повторное размытие  $\sigma_0 = 1$  и используем детектор края Canny [9] для выполнения обнаружения края. Шкала размытия оценивается в каждом краевом положении, образуя редкую карту глубины, обозначаемая  $\hat{d}(x)$ . Однако из-за шума или мягких теней оценки размытия могут содержать некоторые ошибки. Чтобы подавить эти ошибки, применяется совместный двусторонний фильтр [10] на разреженной карте глубин  $\hat{d}(x)$ . Выход совместного двустороннего фильтра в каждом краевом положении x определяется как:

$$BF(\hat{d}(x)) = \frac{1}{W(x)} \sum_{y \in N(x)} G_{\sigma_s}(\|x - y\|) G_{\sigma_r}(\|I(x) - I(y)\|) \hat{d}(y) \quad (5)$$

где  $W(x)$  - коэффициент нормировки, а  $N(x)$  - окрестность точки x, заданная размером пространственного гауссовского фильтра  $G_{\sigma_s}$ . Фильтрация выполняется только по краям. Совместный двусторонний фильтр корректирует некоторые ошибки в разреженной карте глубины и избегать распространения этих ошибок в интерполяции глубины, описанной в следующем разделе.

#### 1.4. Интерполяция глубины

Данный метод оценки размытия дает разреженную карту глубин  $d(x)$  с оценками глубины в местах краев. Чтобы получить полную карту глубины  $d(x)$ , необходимо распространить эти значения из местоположений краев на

все изображение. Найдем регуляризованное отображение глубины  $d(x)$ , которое близко к разреженной карте глубин  $d(x)$  в каждом краевом положении. Для этих задач обычно используются методы интерполяции с учетом края [11]. Здесь применяется матирующий лапласиан [12] для выполнения интерполяции. Мы переписываем разреженную карту глубин  $d(x)$  и полное отображение глубины  $d(x)$  в их векторной форме как  $d$  и  $d$ . Тогда проблему интерполяции глубины можно сформулировать как минимизирующую следующую функцию стоимости:

### 1.5. Тестирование стабильности и качества алгоритма

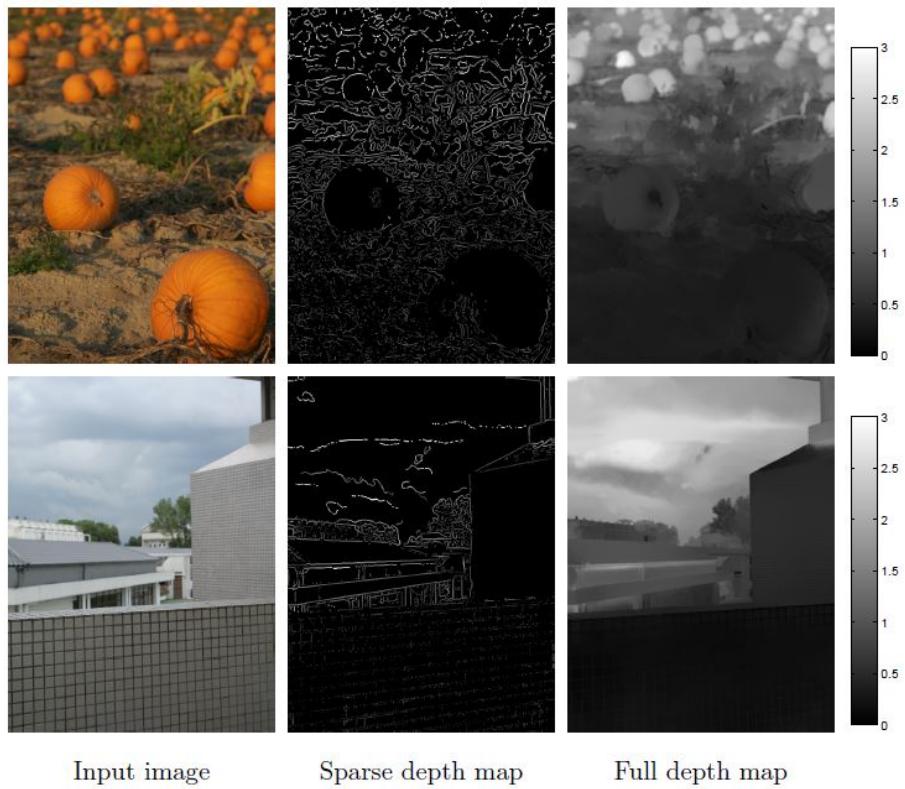


Рисунок 5 – Восстановление глубины на реальных изображениях. Наш метод может работать как на сценах с непрерывной глубиной (изображение тыквы), так и на слоистой глубине (изображение здания) для получения карты глубины с довольно хорошей точностью.

Как показано на рисунке 5, тестируем наш метод на некоторых реальных изображениях. В изображении тыквы глубина сцены непрерывно изменяется от нижней к верхней части изображения. Оценочная карта глубины фиксиру-

ет непрерывное изменение глубины. В изображении здания сцена в основном содержит три слоя: стены, дом и небо. Наш метод позволяет создавать карты глубин, точно представляющие эти слои глубины. Как видно из результатов, наш метод позволяет точно восстановить глубину сцены из одного расфокусированного изображения. На рисунке 6 мы сравниваем наш метод с методом обратной диффузии [7].

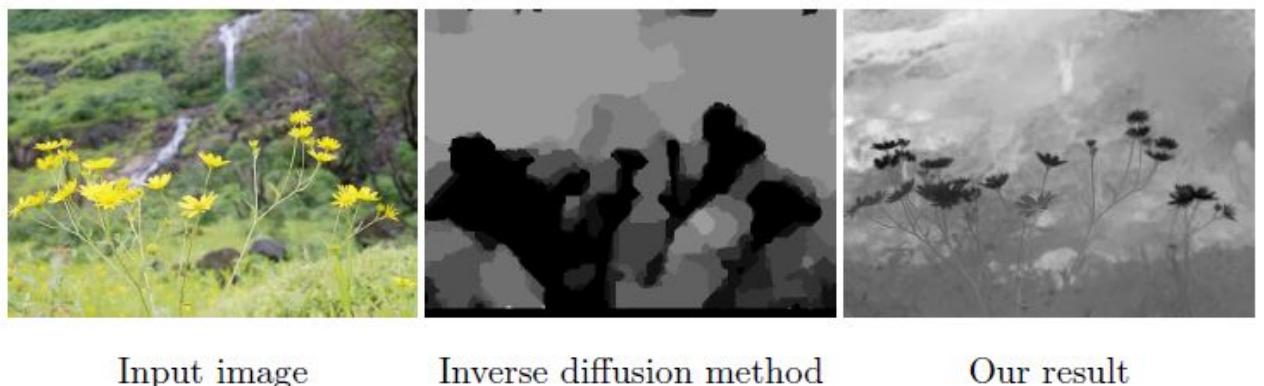


Рисунок 6 – Сравнение нашего метода с методом обратной диффузии, на примере цветка

Метод обратной диффузии создает грубую слоистую карту глубины. В результате этого цветочный слой плохо отделен фоновыми слоями и содержит некоторые оценки погрешности. Напротив, наш метод способен производить более точную и непрерывную карту глубины, а цветочный слой хорошо отделен фоновой травой и деревьями.

## 1.6. Оценка достоверности построения карты глубины

В ходе выполнения проекта будет создана методика количественной оценки достоверности построения карты глубины, которая необходима для корректного преобразования 2D изображения в 3D вид. Эта методика основана на сравнении показаний глубины, полученных в результате работы алгоритмов преобразования 2D  $\rightarrow$  3D и достоверных показаний глубины для данной сцены, которые получены в ручном режиме.

Для реализации методики предполагается произвести ручную разметку тестовых цифровых фотографий в количестве не менее 100 штук. Для этого в ручном режиме устанавливаются контрольные точки, с указанием рассто-

яния до них.

Таким образом формируется разреженная карта глубины, с указанием абсолютных значений. Далее, на изображении находится самая дальняя контрольная точка и все расстояния нормируются на расстояние до нее. Таким образом формируется разреженная карта глубины с относительными расстояниями, и, как следствие, (алгоритмически) формируется «эталонная» плотная карта глубины.

Далее, в результате работы различных алгоритмов преобразования изображения из двумерного в трехмерное автоматически вычисляется как разреженная карта глубины для каждого из анализируемых алгоритмов (создаваемого в рамках проекта и конкурирующих аналогов). Далее предполагается вычислять среднеквадратичную ошибку между эталонными значениями для карты глубины и значениями, вычисленными каждым из алгоритмов.

Предполагается, что алгоритм, реализуемый в ходе выполнения проекта даст снижение такой ошибки между вычисленными и эталонными значениями на величину не менее 25% (ожидаемое значение – порядка 30%).

## **2. Описание предметной области**

### **2.1. Android Studio**

Android Studio - полностью укомплектованная платформа для разработки и тестирования приложений под операционную систему Android. Разработчики этой оболочки (компания Google) внедрили весь необходимый инструментарий для удобного и качественного проектирования новых приложений и доработки существующих. Программа включает в себя такие компоненты как Android SDK, все версии операционки Android, эмулятор для запуска приложений, элементы тестирования и отладки программ.

Создавая новый проект, будет доступна полная структура приложения со всеми файлами, что позволяет более четко и продуманно организовать сам процесс разработки. Очень удобно реализован показ вносимых изменений и дополнений - визуально в реальном времени происходят преобразования в зависимости от заданных действий. Что немаловажно, программа позволяет делать разработку приложений для всех версий ОС Andriod и для различных устройств - можно предварительно оценить внешний вид программы, например, под планшет или смартфон.

Среда Android Studio предназначена как для небольших команд разработчиков мобильных приложений (даже в количестве одного человека), или же крупных международных организаций с GIT или другими подобными системами управления версиями. Опытные разработчики смогут выбрать инструменты, которые больше подходят для масштабных проектов. Решения для Android разрабатываются в Android Studio с использованием Java или C++. В основе рабочего процесса Android Studio заложен концепт непрерывной интеграции, позволяющий сразу же обнаруживать имеющиеся проблемы. Продолжительная проверка кода обеспечивает возможность эффективной обратной связи с разработчиками. Такая опция позволяет быстрее опубликовать версию мобильного приложения в Google Play App Store. В Android Studio есть удобная маркировка кода, которая позволит без труда ориентироваться в больших проектах. Кроме того, отдельные компоненты можно изменять простым перетаскиванием в другое нужное место, что значительно упрощает редактирование.

## **2.2. Java**

Преимущества языка Java

- Одно из основных преимуществ языка Java — независимость от платформы, на которой выполняются программы: один и тот же код можно запускать под управлением операционных систем Windows, Solaris, Linux, Macintosh и др. Это действительно необходимо, когда программы загружаются через Интернет для последующего выполнения под управлением разных операционных систем.
- Другое преимущество заключается в том, что синтаксис языка Java похож на синтаксис языка C++, и программистам, знающим языки С и C++, его изучение не составляет труда.
- Кроме того, Java — полностью объектно-ориентированный язык, даже в большей степени, чем C++. Все сущности в языке Java являются объектами, за исключением некоторых основных типов (primitive types), например чисел.
- Исключена возможность явного выделения и освобождения памяти. Память в языке Java освобождается автоматически с помощью механизма сборки мусора. Программист гарантирован от ошибок, связанных с неправильным использованием памяти.
- Безопасный: методы проверки подлинности основаны на шифровании с открытым ключом.
- Динамический: программирование на Java считается более динамичным, чем на С или C++, так как он предназначен для адаптации к меняющимся условиям. Программы могут выполнять обширное количество во время обработки информации, которая может быть использована для проверки и разрешения доступа к объектам на время выполнения.

## **2.3. Технические требования.**

Необходимая задача при реализации проекта - адаптация существующего метода преобразования 2D в 3D для использования под управлением опера-

ционной системы Android.

В ходе выполнения проекта будут созданы мобильные приложения со следующими характеристиками:

- мобильное приложение, функционирующее под управлением операционной системы Android версии не ниже 5.0.
- специальных дополнительных требований к аппаратному обеспечению мобильных устройств не предъявляется. Они соответствуют требованиям, накладываемым вышеперечисленными версиями ОС.
- мобильные приложения должны функционировать на устройствах с разными размерами экрана от 2.6 до 6 дюймов с разрешениями от 240x320 до 1440x2560 пикселей.
- источник данных для конвертирования - файлы в формате jpg из галереи мобильного устройства или фотография, сделанная штатной фотокамерой мобильного устройства.
- мобильные приложения должны обеспечивать конвертацию сделанных фотокамерой изображений при максимальном качестве съемки не менее 10 Мп.
- формат данных, в которых сохраняются результаты — аниглиф файлы (формат jpeg), стерео пара (формат jps), анимированные файлы в формате gif.
- ориентированное время преобразования 2D в 3D не должно превышать 5 секунд.

## **2.4. Изучение основ разработки мобильного приложения в среде Android Studio**

Android основан на Linux. Между приложением и ядром лежит слой API и слой библиотек на нативном коде. Приложение выполняется на виртуальной машине Java (Dalvik Virtual Machine). В Android можно запускать много приложений. Но одно из них есть главным и занимает экран. От текущего

приложения можно перейти к предыдущему или запустить новое. Это похоже на браузер с историей просмотров.

Каждый экран пользовательского интерфейса представлен классом Activity в коде. Различные Activity содержатся в процессах. Activity может даже жить дольше процесса. Activity может быть приостановлена и запущена вновь с сохранением всей нужной информации.(рисунок 7)

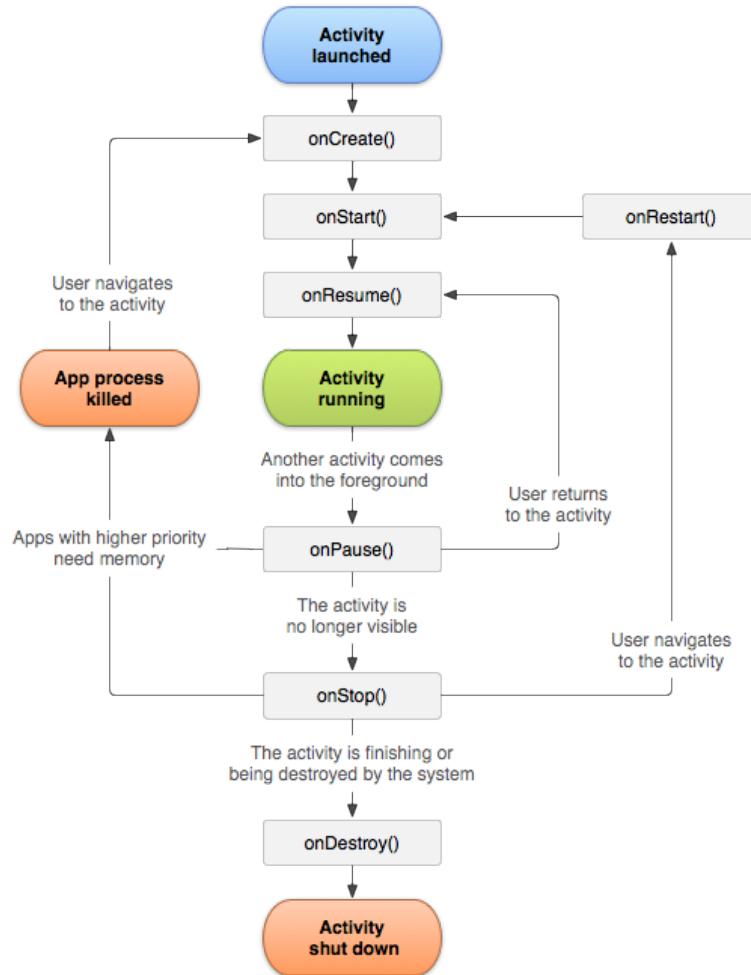


Рисунок 7 – activity

Android использует специальный механизм описания действий основанный на Intent. Когда нужно выполнить действие (сделать звонок, послать письмо, показать окно), вызывается Intent.

Также Android содержит сервисы подобные демонам в Linux для выполнения нужных действий в фоновом режиме (например, проигрывание музыки). Для обмена данными между приложениями используются Content providers (провайдеры содержимого).

Содержимое Activity формируется из различных компонентов, называемых View. Самые распространенные View - это кнопка, поле ввода, чекбокс и т.д. (рисунок 8)

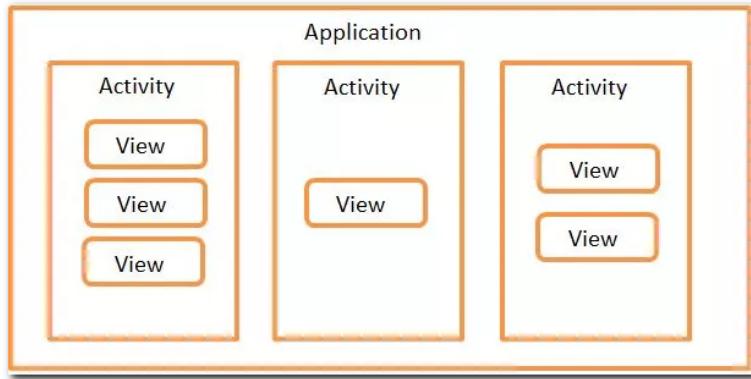


Рисунок 8 – view

Необходимо заметить, что View обычно размещаются в ViewGroup. Самый распространенный пример ViewGroup – это Layout. Layout бывает различных типов и отвечает за то, как будут расположены его дочерние View на экране.

LinearLayout – отображает View-элементы в виде одной строки (если он Horizontal) или одного столбца (если он Vertical).

TableLayout – отображает элементы в виде таблицы, по строкам и столбцам.

RelativeLayout – для каждого элемента настраивается его положение относительно других элементов.

AbsoluteLayout – для каждого элемента указывается явная позиция на экране в системе координат (x,y)

## 2.5. Основные компоненты пользовательского интерфейса мобильного приложения в Android

В Android используется UI-фреймворк, сравнимый с другими полнофункциональными UI-фреймворками, применяемыми на локальных компьютерах. Он является более современным и асинхронным по природе. По существу, UI-фреймворк Android относится уже к четвертому поколению, если считать первым поколением традиционный прикладной интерфейс программирования Microsoft Windows, основанный на C, а MFC (Microsoft Foundation

Classes, библиотека базовых классов Microsoft на основе C++) - вторым. В таком случае UI-фреймворк Swing, основанный на Java, будет третьим поколением, так как предлагаемые в нем возможности дизайна значительно пре- восходят по гибкости MFC. Android UI, JavaFX, Microsoft Silverlight и язык пользовательских интерфейсов Mozilla XML (XUL) относятся к новому типу UI-фреймворков четвертого поколения, в котором UI является декларативным и поддерживает независимую тематизацию.

При программировании в пользовательском интерфейсе Android применяется объявление интерфейса в файлах XML. Затем эти определения представления (view definitions) XML загружаются в приложение с пользовательским интерфейсом как окна. Даже меню приложения загружаются из файлов XML. Экраны (окна) Android часто называются активностями (activities), которые включают в себя несколько видов, нужных пользователю, чтобы выполнить логический элемент процесса. Виды (views) являются основными элементами, из которых в Android состоит пользовательский интерфейс. Виды можно объединять в группы (view groups). Для внутренней организации видов используются давно известные в программировании концепции холст (canvas), рисование (painting) и взаимодействие пользователя с системой (user interaction).

Такие составные представления, в которые входят виды и группы видов, работают на базе специального логического заменяемого компонента пользовательского интерфейса Android.

Одной из ключевых концепций фреймворка Android является управление жизненным циклом (lifecycle) окон явлений (activity windows). В системе применяются протоколы, поэтому Android может управлять ситуацией по мере того, как пользователи скрывают, восстанавливают, останавливают и закрывают окна явлений.

### **3. Анализ существующих продуктов со схожим функционалом**

В ходе подготовки к процессу разработки приложения был проведен анализ магазина мобильных приложений (Google Play) для проверки на наличие аналогичных разработок или разработок схожей тематики.

По результатам анализа оказалось, что такой реализации преобразования фотографий в 3D нет. Однако в этой категории есть ряд похожих разработок, направленных на оживление «плоских» снимков.

#### **3.1. Make It 3d Free**

Приложение, позволяющее создать красно-синий анагlyph либо на основе снимка из фотогалереи смартфона, либо сделав снимок прямо из приложения. Стоит заметить, что анагlyph создается на основе двух снимков: левого и правого, - что само по себе является естественным способом создание объемного изображения, а значит должно гарантировать получение качественного 3D-изображения. После прикрепления двух изображений доступны возможности изменения стереоэффекта: смещение или поворот одного из изображений по или против часовой стрелки. Есть возможность сохранения результата в виде широкого изображения с разделением зон на правый и левый глаз, что может пригодиться для просмотра снимка в устройстве виртуальной реальности (например, в кардборде). По завершении работы, фото экспортируется в галерею и открывается стандартном просмотрщике изображений девайса. (рисунок 9)

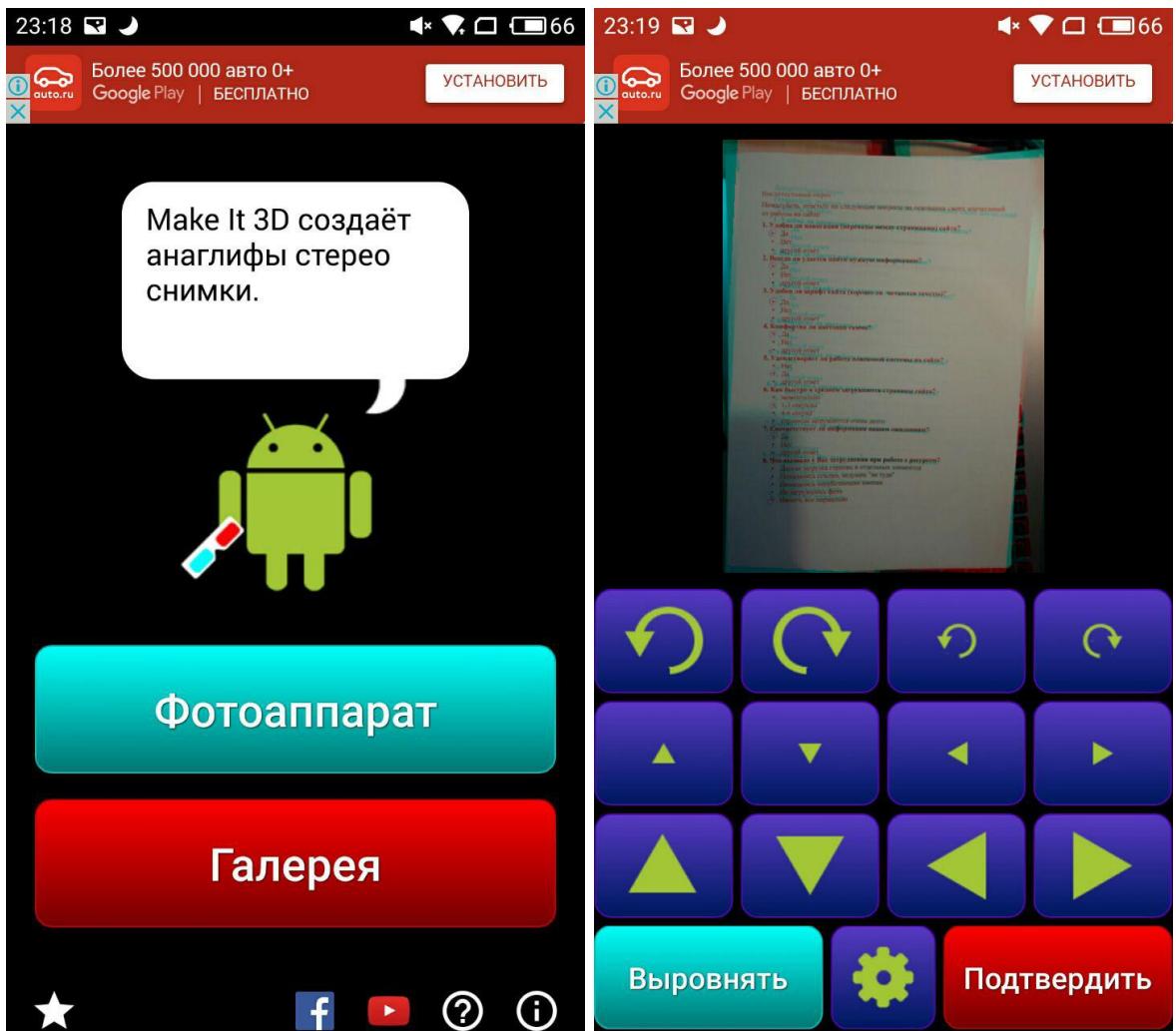


Рисунок 9 – MakeIt

### 3.2. 3D Effect

Приложение, создающее анаглиф из одного изображения. Доступна возможность настройки степени смещение и выбор одного из трех вариантов смещения: по горизонтали, по вертикали, по диагонали. Так предоставлена возможность выбора цвета стереоэффекта, например: красно-голубой, сине-желтый, зелёно-розовый. Изображение сохраняется в фотогалерею смартфона и выводит варианты «шэринга». (рисунок 10)

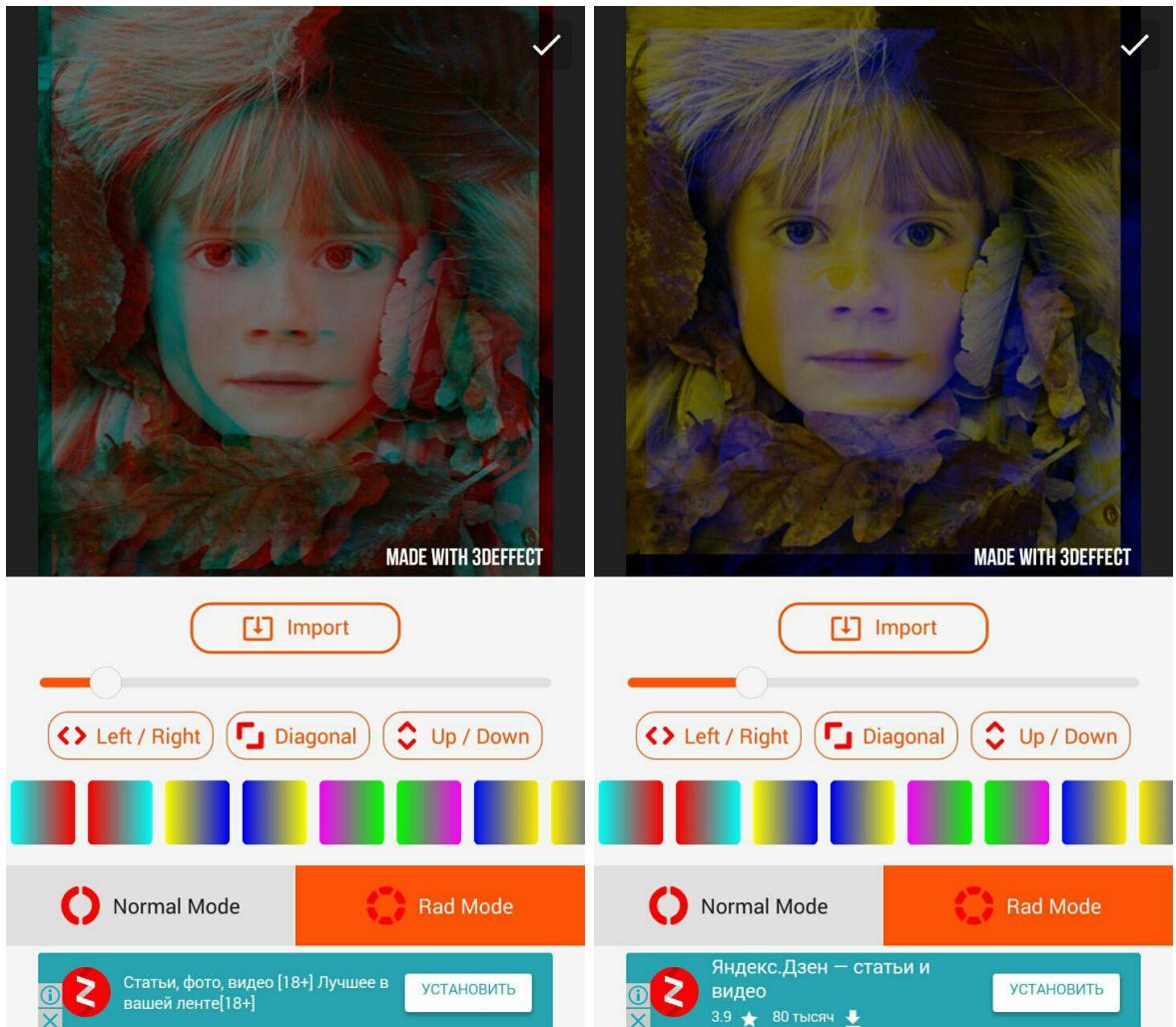


Рисунок 10 – 3dEffect

### 3.3. Motion Photo

Инструмент, предоставляемый производителями смартфонов. Впервые появился в iOS 9 в 2015 году под названием Live Photos, а позже аналоги появились в Android-смартфонах, линейке Galaxy от Samsung, начиная с модели S7, а также в Google Pixel 2. Решение в корне отличается от варианта с созданием анаглифа, однако предоставляет возможность оживить фотографию, что так же является и результатом работы нашей разработки. Принцип работы заключается в постоянном сохранении в оперативную память устройства 1-3 секунд изображения с камеры, до момента нажатия на кнопку спуска затвора, после чего сохраняется еще один видеофрагмент. Таким образом, при просмотре обычного фото в галерее смартфона, появляется возможность увидеть живой момент съемки.

### 3.4. Fyuse

Fyuse совмещает в себе возможность создания пространственной фотографии и платформу для общения. По ходу использования приложения фокусируется на объекте, а затем при помощи графических подсказок на экране предлагает обойти объект камерой по орбите. Полученный результат представляет собой снимок, позволяющий перемещать камеру вида по орбите вокруг изображенного предмета при помощи свайпов или гироскопа смартфона. Фотографию можно сохранить или опубликовать прямо во Fyuse, являющейся так же и социальной сетью. (рисунок 11)

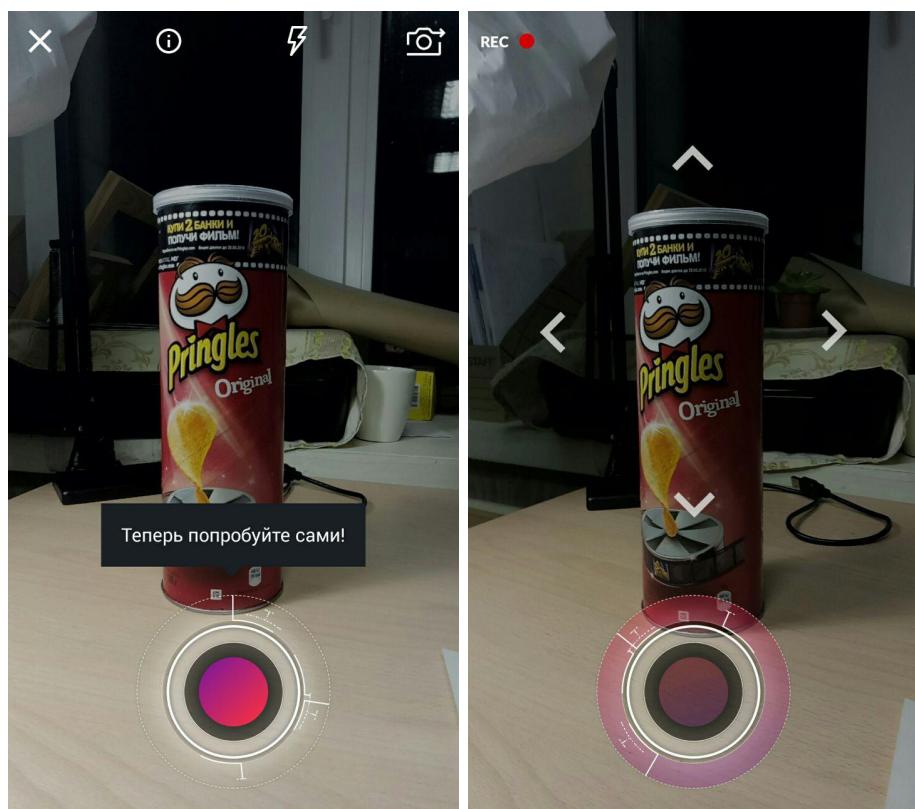


Рисунок 11 – Fyuse

### 3.5. Вывод

По результатам анализа существующих аналогов можно сделать вывод, что разработчики предлагают абсолютно разные способы оживления снимков. Для некоторых требуются определенный манипуляции еще при съемке, некоторые могут работать с готовыми фотографиями. Каждое решение по-своему уникально и интересно. Мы же предлагаем еще одно решение, особенностью которого является «оживление» уже готового снимка.

## **4. Разработка пользовательского интерфейса мобильного приложения (UI/UX)**

Одна из важнейших задач в ходе создания мобильного приложения, преобразовывающего 2D снимки в объемные (3D) - разработка удобного и интуитивно-понятного пользовательского интерфейса. UI/UX (user Interface, user experience) составляющая, она же пользовательский интерфейс и пользовательский опыт, является более чем просто значимым элементом современного мобильного приложения. Удобство расположения элементов управления и приятное визуальное оформление напрямую влияют на настроение пользователя при использовании продукты. Именно некачественный UI/UX-дизайн отпугивает людей от использования многих приложений в пользу их более достойных альтернатив.

### **4.1. Анализ интерфейсов смежных приложений**

Для использования в качестве референсов были подобраны приложения, принадлежащие широкому понятию «Приложения для работы с фото», так как узкоспециализированные решения не представляют интерфейса должного качества и удобства.

В список приложений для анализа попали такие приложения, как: Prisma, Vinci, Snapster и Pixlr.

В первую очередь, целью сравнения являлся анализ особенностей пользовательского интерфейса этих приложений. Это обусловлено тем, что предполагаемая схема взаимодействия пользователя с разрабатываемым приложением во много схожа с таковой в этих приложениях, за исключением некоторых их особенностей. Важные элементы для сравнения - сценарий создания снимка с использованием фотокамеры смартфона и выбор снимка из галереи для дальнейшей отправки фотографии на обработку алгоритмом, преобразующим ее в 3D фотографию.

#### **4.1.1. Prisma**

Проанализировав пользовательский интерфейс первого приложения, Prisma (рисунок 12), можно заметить не очень удобное расположение элементов. Так,

кнопка настроек, далеко не самая востребованная на фоне остальных, расположена в правом нижнем углу, куда доступ наиболее прост в условиях больших размеров современных смартфонов. В том время как кнопка смена камеры расположена в правом верхнем углу, куда доступ затруднительнее. Стоит отметить и факт смещения нижних иконок в углу экрана, что далеко не положительном образом сказывается на опыте использования приложения.

Также Prisma, в связи со своими социальными особенностями функционала, имеет навигационный бар в верхней части экрана, представленный фиксированными вкладками (fixed tabs), оформленным в соответствии с гайдлайнами material-дизайна (Material Design Guidelines), разработанными Google. В целом приложение лишь частично следует гайдлайнам от Google, что заметно на примере применения material-иконок и одновременно неверном использовании фаба (float action button).

Экран шейринга фотографии выполнен по примеру Instagram, с подобным размещением элементов и учетом особенностей Prisma.

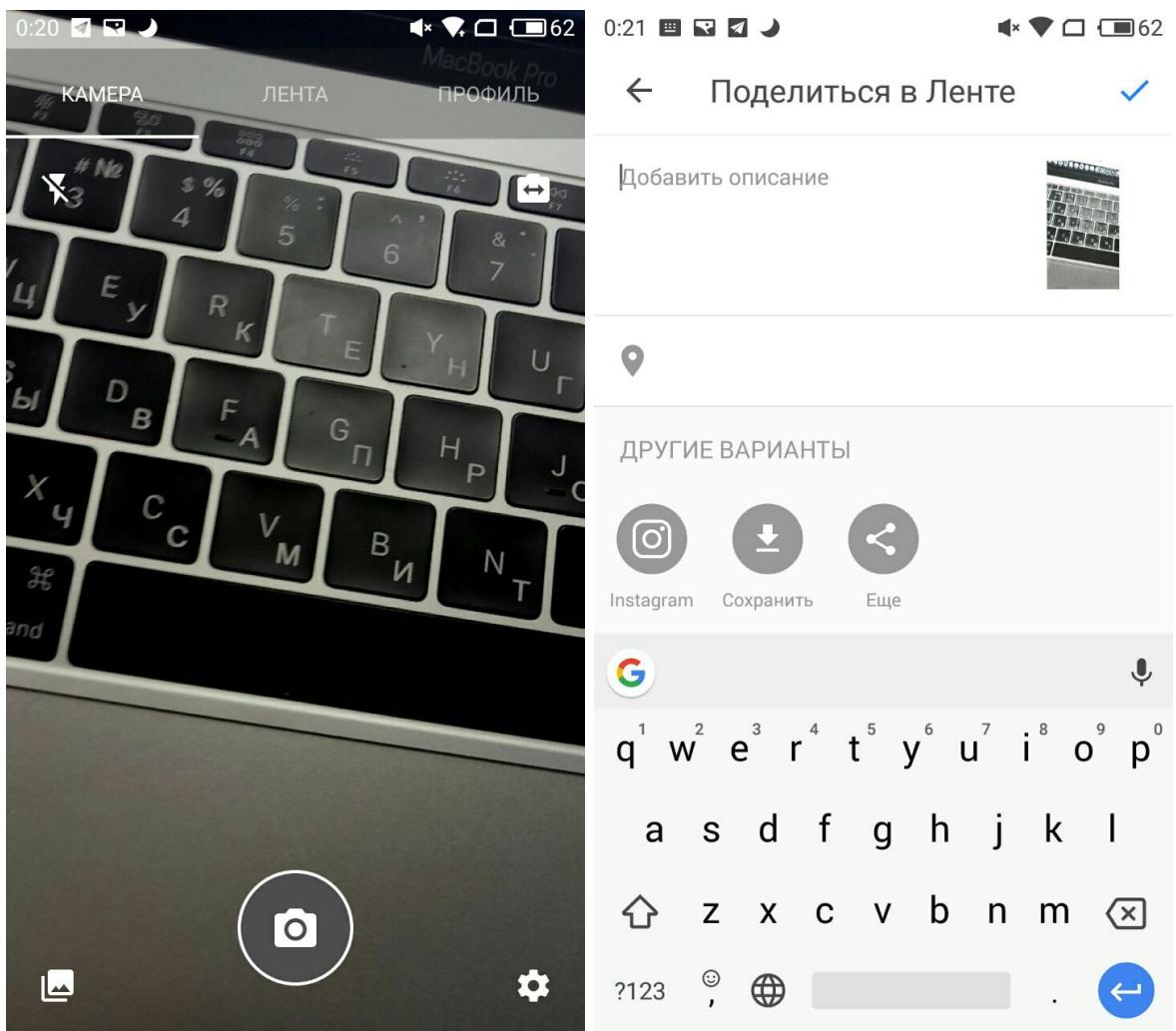


Рисунок 12 – Prisma

#### 4.1.2. Vinci

Интерфейс Vinci (рисунок 13) выглядит более удачным, на фоне Prisma, и, забегая вперед, на фоне остальных выбранных приложений также, за исключением Snapster. В нижней части удачно разместилась выдвигающаяся панель с последними снимками, выше кнопка спуска затвора, включения вспышки и смены камеры. В верхнем углу разместилась иконка настроек. Такое разделение экрана обусловлено популярным соотношением сторон для снимков 4:3.

Плашка шейринга, появляющаяся в нижней части экрана, также весьма удобна для использования.

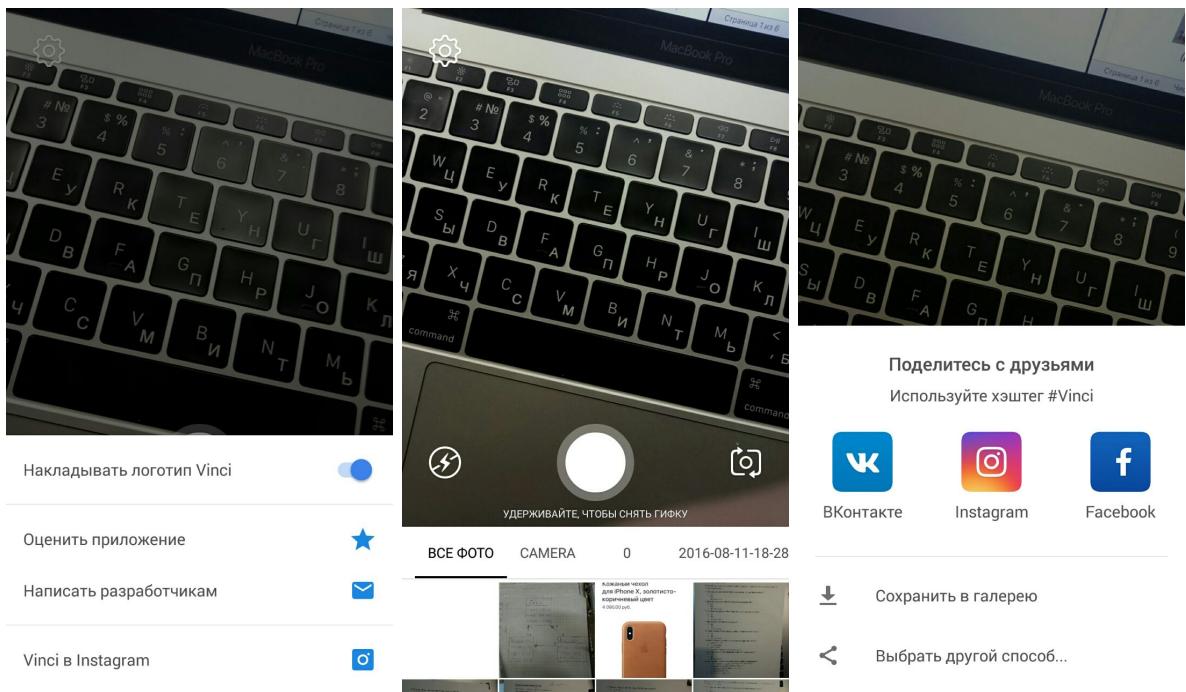


Рисунок 13 – Vinci

#### 4.1.3. Pixlr

Особенностью интерфейса Pixlr (рисунок 14), в отличие от подобанных примеров, является присутствие стартового экрана, который, однако, несколько замедляет взаимодействие с приложением. Кнопки управление на экране камеры визуально неприятны и имеют непривычное соотношение размеров.

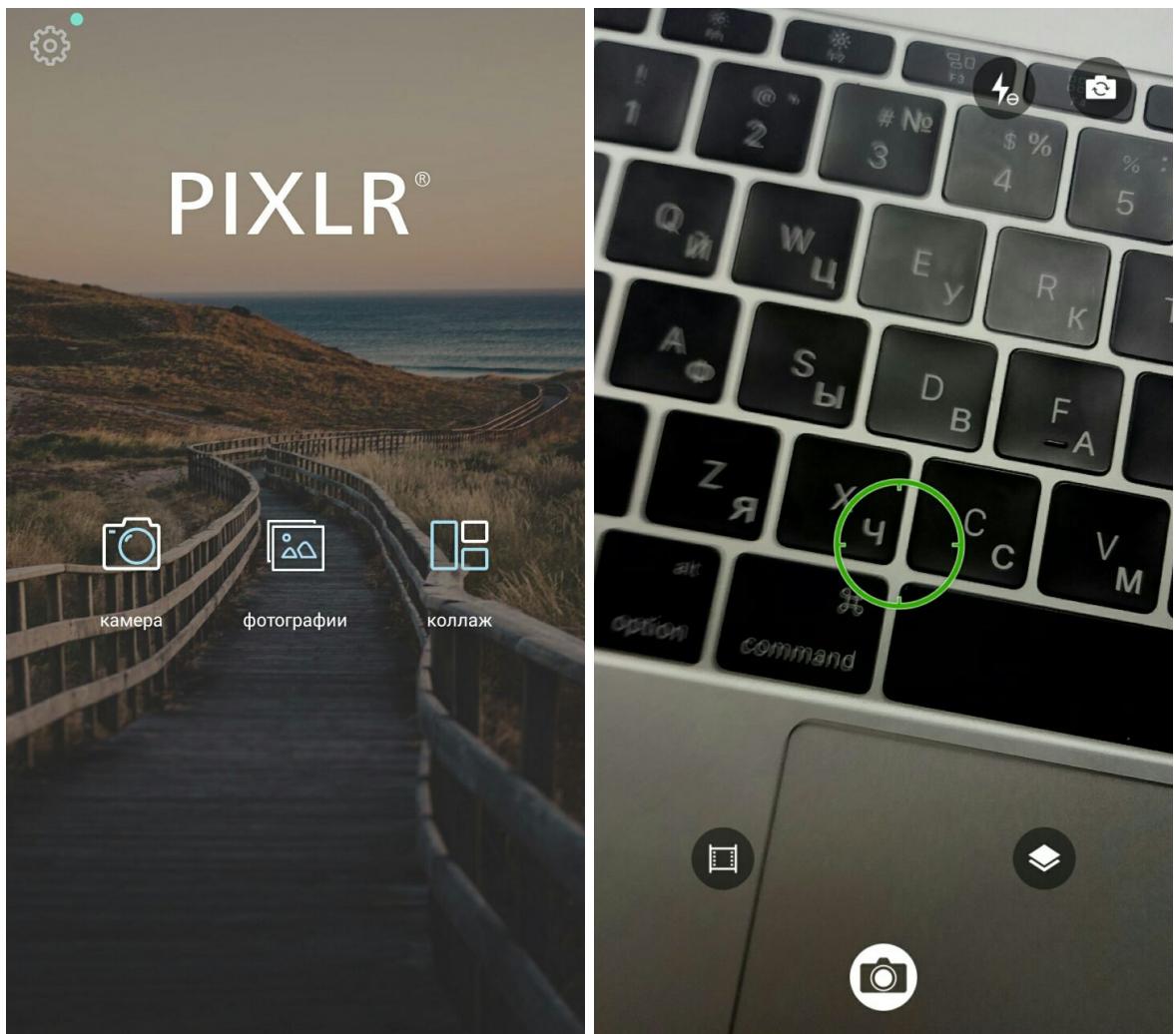


Рисунок 14 – Pixlr

#### 4.1.4. Snapster

Как уже было сказано, Vinci и Snapster (рисунок 15) имеют примерно одинаковый уровень юзабилити, что объясняется одним и тем же авторством. В нижней части экрана размещен блок с последними снимками, но уже другим способом навигации - горизонтальной прокруткой (скроллом). Разработчик решил не внедрять кнопку настроек, нашлось место для полезной кнопки активации сетки. В целом, приложение Snapster, являясь экспериментальным проектом социальной сети VK, претерпело за время своего существования довольно сильные изменения. Если в первоначальном виде приложение дублировало основной функционал модуля «Фотографии» из социальной сети VK и расширяло его возможности, то сейчас, не сыскав популярности, было урезано до простенького фоторедактора.

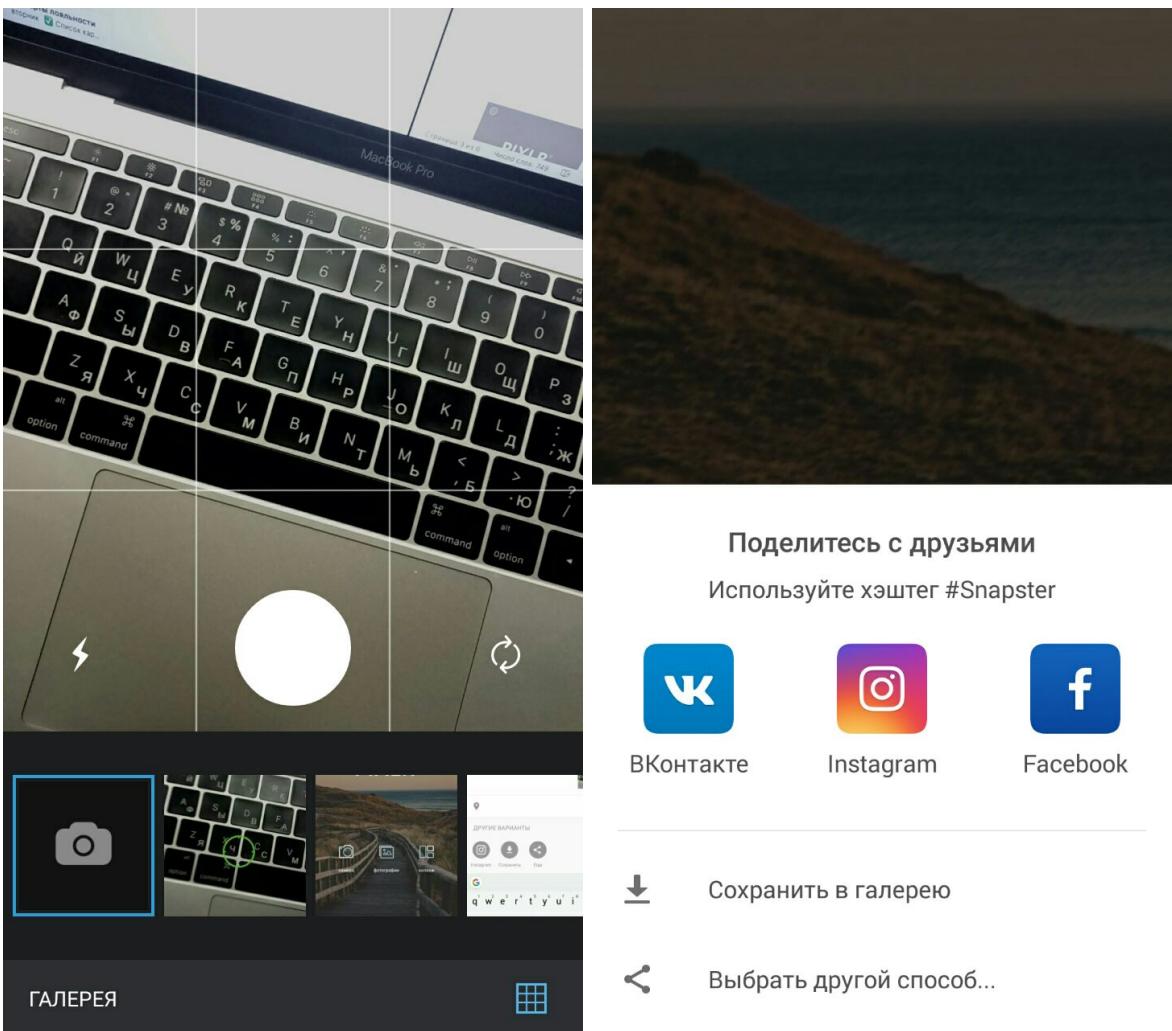


Рисунок 15 – Snapster

## 4.2. Разработка пользовательского интерфейса мобильного приложения (UI/UX)

Интерфейс Android-смартфона мобильного приложения для преобразования 2D фотографии в 3D вид выполнен в соответствии с гайдлайнами Material Design. Отступы, иконки и используемые шрифты полностью соответствуют руководству от Google [5].

Расположение элементов управления выполнено с учетом лучших особенностей приложений-аналогов.

На главный экран камеры (рисунок 18) выведены следующие функции:

- Спуск затвора;
- Выбор фото из галереи;

- Смена камеры;
- Управление вспышкой;
- Переход к настройкам и справке.

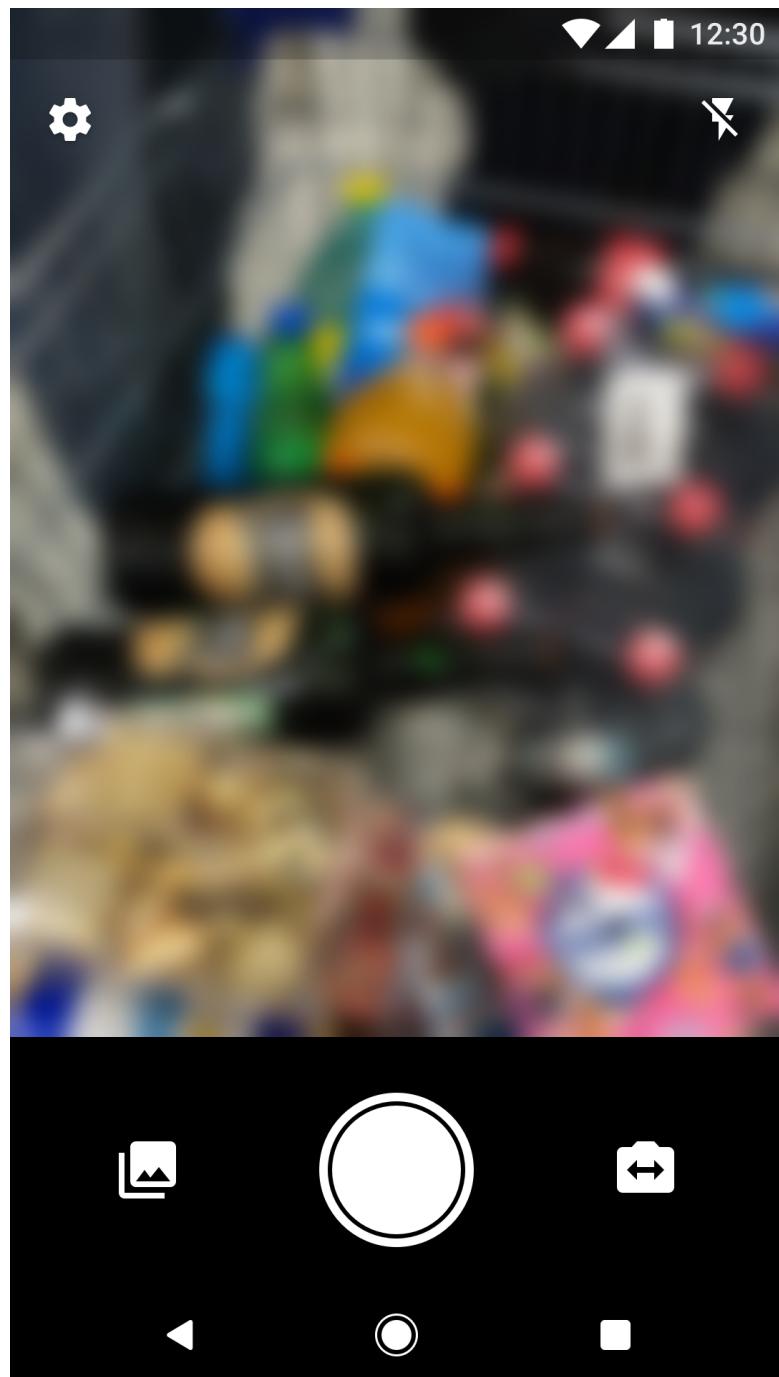


Рисунок 16 – Окно с камерой

На экране с обработанной фотографией по нажатии кнопки «Далее» появляется bottom sheet (рисунок 19), включающий в себя быстрые функции шеринга и сохранения полученного фото в галерею.

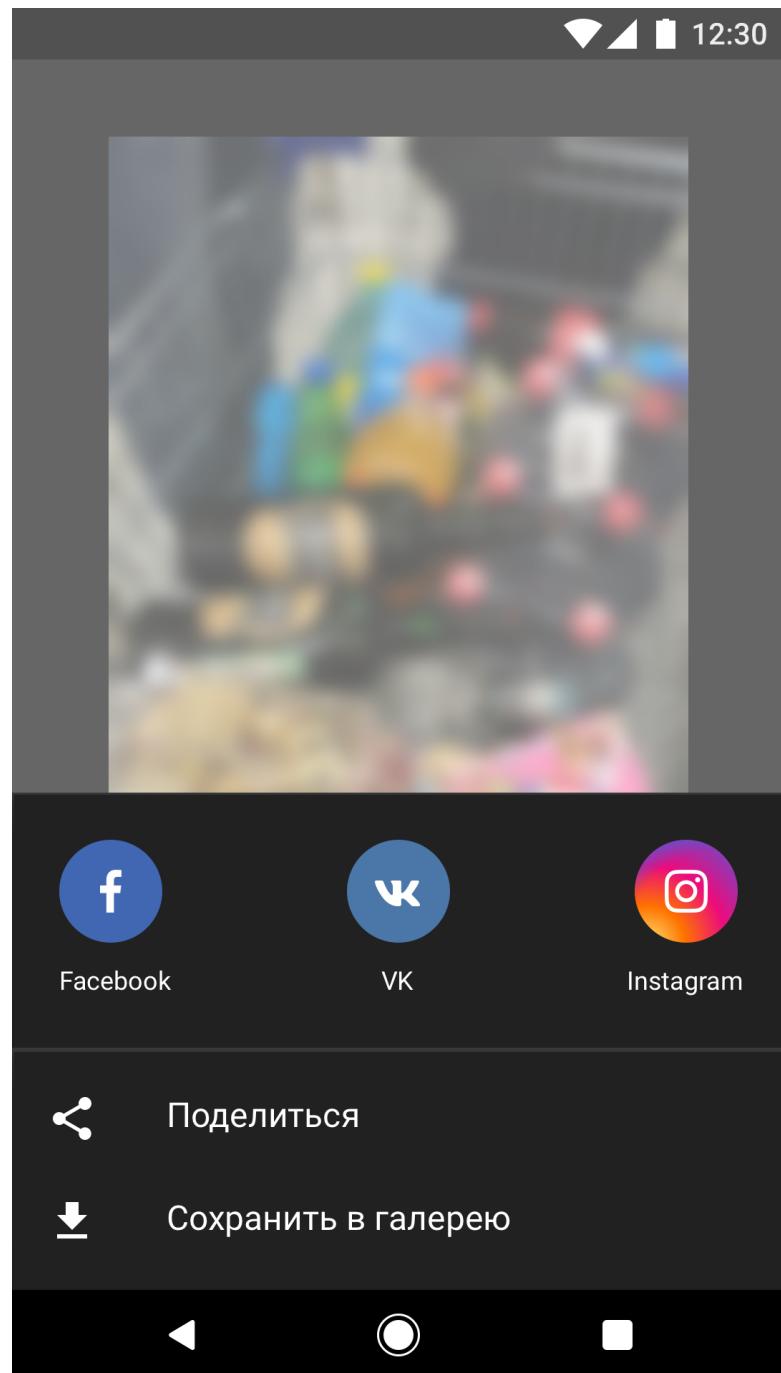


Рисунок 17 – Окно с просмотром фото

При создании пользовательского интерфейса приложения были проанализированы современные, с аналогичным функционалом мобильные приложения в целом. Был сделан акцент на необходимости создания современного, функционального и не перегруженного пользовательского интерфейса. В результате проведенного анализа был создан пользовательский интерфейс мобильного приложения для преобразования 2D фотографий в 3D вид.

### **4.3. Проектирование структуры приложения**

Одна из важнейших задач в ходе создания мобильного приложения, преобразовывающего 2D снимки в объемные (3D) - разработка удобного и интуитивно-понятного пользовательского интерфейса. UI/UX (user Interface, user experience) составляющая, она же пользовательский интерфейс и пользовательский опыт, является более чем просто значимым элементом современного мобильного приложения. Удобство расположения элементов управления и приятное визуальное оформление напрямую влияют на настроение пользователя при использовании продукты. Именно некачественный UI/UX-дизайн отпугивает людей от использования многих приложений в пользу их более достойных альтернатив.

На главный экран камеры (рисунок 18) выведены следующие функции:

- Спуск затвора;
- Выбор фото из галереи;
- Смена камеры;
- Управление вспышкой;
- Переход к настройкам и справке.

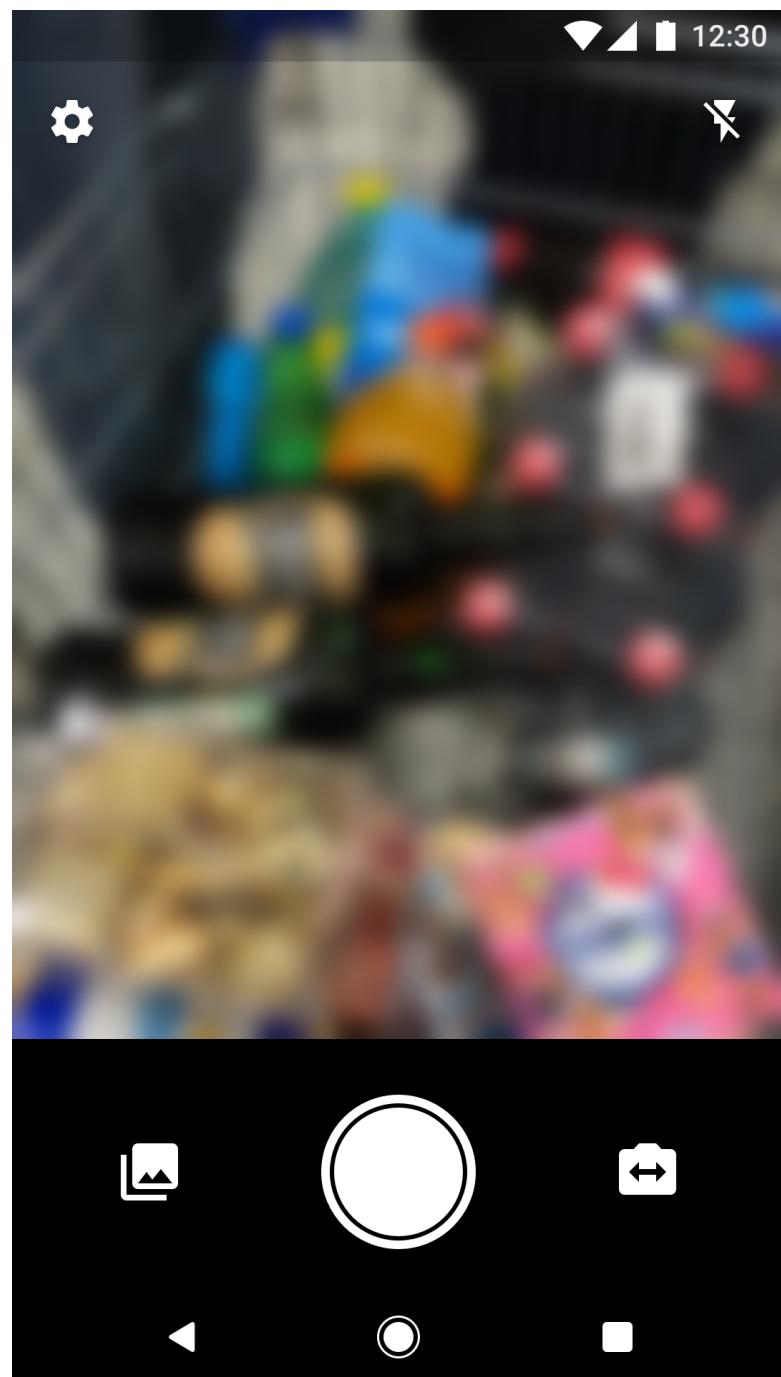


Рисунок 18 – Окно с камерой

На экране с обработанной фотографией по нажатии кнопки «Далее» появляется bottom sheet (рисунок 19), включающий в себя быстрые функции шеринга и сохранения полученного фото в галерею.

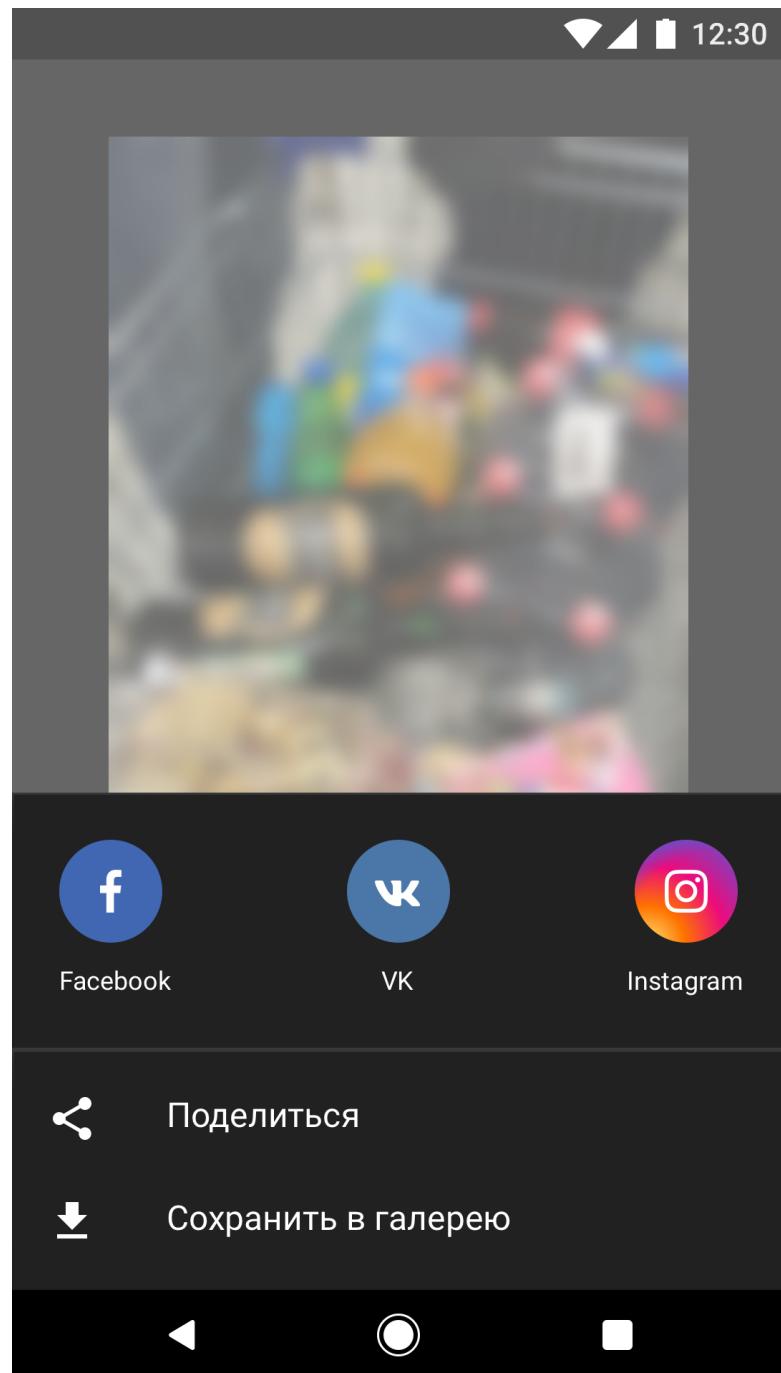


Рисунок 19 – Окно с просмотром фото

При создании пользовательского интерфейса приложения были проанализированы современные, с аналогичным функционалом мобильные приложения в целом. Был сделан акцент на необходимости создания современного, функционального и не перегруженного пользовательского интерфейса. В результате проведенного анализа был создан пользовательский интерфейс мобильного приложения для преобразования 2D фотографий в 3D вид.

## **5. Разработка мобильного приложения**

### **5.1. Разработка концепции и архитектуры мобильного приложения, предназначенных для преобразования 2D в 3D**

В ходе выполнения работы были рассмотрены различные варианты для создания мобильных приложений, предназначенных для преобразования 2D изображений в 3D вид. При этом рассмотрении учитывалось, что результирующие мобильные приложения должны создаваться под операционные системы Android и iOS, а также то, что один из основных результатов работы приложения с точки зрения конечного пользователя – это возможность публикации созданного 3D-изображения в виде анимированного gif-файла в одном или нескольких аккаунтах в социальных сетях пользователя. Соответственно, можно исходить из предположения о том, что для функционирования приложения в любом случае необходим доступ к сети интернет. Максимальная унификация различных составных частей приложения между собой хотя бы на уровне исходных кодов вне зависимости от целевой платформы (Android или iOS) является дополнительным преимуществом при рассмотрении различных вариантов создания мобильных приложений.

Один из наиболее простых с технической точки зрения вариантов реализации решения, позволяющего преобразовывать 2D файлы в 3D вид, является решение, основанное на создании веб-сервиса, который предоставляет минимально необходимый пользовательский интерфейс для загрузки желаемого файла на сервер, преобразования файла на сервере и, как результат, возможность скачать получившийся файл на устройство пользователя и поделиться этим файлом в социальных сетях. При простоте архитектуры у этого решения есть один существенных недостаток – как правило, такие решения менее удобны и функциональны, чем нативные (native) мобильные приложения, разработанные специально под целевую платформу, на которой они будут функционировать. Поэтому на данном этапе от такого подхода было принято решение отказаться.

Если рассматривать варианты создания нативных мобильных приложений, существует несколько возможных вариантов:

- 1) Использовать наиболее популярные средства разработки и языки программирования для каждой из необходимых мобильных платформ. Создать нативное мобильное приложение, реализующее весь необходимый пользовательский интерфейс, набор сервисных функций. Портировать алгоритм преобразования графического файла из 2D в 3D для локального исполнения на мобильном устройстве. Все необходимые преобразования выполнять локально, на мобильном устройстве. Полученный результат преобразования (анимированный gif) загружать в интернет (социальные сети) по мере его готовности на мобильном устройстве.
- 2) Использовать наиболее популярные средства разработки и языки программирования для каждой из необходимых мобильных платформ для создания нативных мобильных приложений только для реализации пользовательского интерфейса и набора сервисных функций. Алгоритм преобразования графического файла из 2D в 3D реализуется в виде серверного модуля, соответственно для преобразования выбранного файла и предварительного просмотра полученных результатов необходимо загрузить этот выбранный файл на сервер. Загрузить полученный результат с сервера и поделиться этим результатом в социальных сетях.
- 3) Использовать кроссплатформенное средство разработки мобильных решений Xamarin. Это средство одновременно позволяет создавать iOS, Android и Windows Phone приложения. Реализация алгоритма преобразования графического файла из 2D в 3D модифицируется таким образом, что может быть непосредственно интегрирована в Xamarin-проект. В результате получается единое решение, которое на этапе сборки может быть превращено в полнофункциональное нативное приложение для любой из поддерживаемых платформ — iOS, Android и Windows Phone. Результирующее решение будет преобразовывать графические файлы из 2D в 3D локально на мобильном устройстве, доступ к интернету необходим только для реализации функции «поделиться созданным 3D-файлом в социальных сетях».
- 4) Использовать кроссплатформенное средство разработки мобильных решений Xamarin, но только лишь для создания нативных мобильных при-

ложений, реализующих пользовательский интерфейс и набор сервисных функций. Алгоритм преобразования графического файла из 2D в 3D реализуется в виде серверного модуля, соответственно для преобразования выбранного файла и предварительного просмотра полученных результатов необходимо загрузить этот выбранный файл на сервер. Загрузить полученный результат с сервера и поделиться этим результатом в социальных сетях.

Перед тем, как рассмотреть каждый из приведенных выше вариантов по отдельности, следует отметить, что существующая на сегодняшний день реализация алгоритма преобразования изображений из 2D в 3D вид реализована на языке C++, сборка осуществляется с использованием Microsoft Visual Studio, соответствующее решение имеет существенную зависимость от OpenCV (Open Source Computer Vision Library – это программная библиотека с открытым исходным кодом, реализующая базовые алгоритмы компьютерного зрения, обработки изображений и численных алгоритмов общего назначения. Реализована на C/C++).

Возвращаемся к вышеперечисленным вариантам.

Для варианта №1 для операционной системы Android необходимо:

С использованием Android Studio на языке программирования Java реализовать необходимый пользовательский интерфейс, а также весь необходимый набор сервисных функций. Необходимо адаптировать реализацию алгоритма преобразования графического файла из 2D в 3D для использования под управлением операционной системы Android (реализация на C++). Далее, с использованием механизма The Android Native Development Kit (NDK) необходимо обеспечить вызов кода, написанного на языке C++ из «классического» Android-приложения.

Для разработки приложений для iOS основным средством разработки является Objective-C. Проведенные предварительные исследования показали, что адаптировать реализацию алгоритма преобразования графического файла из 2D в 3D для использования под управлением операционной системы iOS с разумными усилиями не представляется возможным. Соответственно, вариант №1 является неприемлемым для реализации решения для операци-

онной системы iOS.

Для реализации варианта №2 необходимо:

С использованием наиболее популярных средств разработки и языков программирования для каждой из необходимых мобильных платформ (Java для Android и Objective C для iOS) необходимо создать нативные мобильных приложений для реализации пользовательского интерфейса и набора сервисных функций. Эта задача, в целом, является типовой и принципиальных сложностей не вызывает. Алгоритм преобразования графического файла из 2D в 3D следует реализовать в виде серверного модуля, например, для использования под управлением операционной системы Ubuntu. Это обусловлено тем, что Unix-подобные операционные системы имеют существенно более широкое распространение в Web-серверном окружении, чем Windows-сервера. Основная сложность данной задачи – в сборке и корректном использовании OpenCV библиотеки под Linux, однако аналогичная задача сравнительно недавно была решена коллегами для реализации демонстрационного видео-плеера для использования под операционной системой Ubuntu.

Для реализации варианта №3 необходимо адаптировать алгоритм преобразования графического файла из 2D в 3D таким образом, чтобы он мог быть непосредственно интегрирован в Xamarin-проект (разработка на языке программирования C#). Проведенные предварительные исследования показали, что реализовать подобный функционал за разумное время с разумными усилиями не представляется возможным.

Вариант №4 основывается на разработке унифицированного решения для всех поддерживаемых мобильных платформ (iOS, Android и Windows Phone) для реализации пользовательского интерфейса и сервисных функций на языке программирования C#. В части алгоритма преобразования графического файла из 2D в 3D этот вариант полностью аналогичен (точнее даже унифицирован) варианту №2 – должен быть реализован серверный модуль, например, для использования под управлением операционной системы Ubuntu для реализации этого преобразования. И, соответственно, с помощью реализованных на C# функций выбирается файл для преобразования, отправляется на сервер, там преобразовывается, скачивается обратно для предварительного просмотра полученных результатов, а далее реализуется возможность поде-

литься полученным результатом в социальных сетях.

На основе проведенного исследования можно сделать следующий вывод. С точки зрения скорости, легкости и качества реализации наиболее перспективными являются варианты №2 и №4. Однако, вариант №4, в силу кроссплатформенности, унифицированности решения для всех поддерживаемых мобильных платформ является более предпочтительным. Соответственно, на реализации этого варианта и будут сосредоточены усилия на самом первом этапе развития проекта.

Очевидным недостатком подобного решения является существенная его зависимость от скорости и надежности мобильного интернета, а также от доступности конечному пользователю оплаченного траффика. Для обхода этих ограничений предполагается исследовать возможность создания для пользователей ОС Android «самодостаточного» мобильного приложения (вариант №1), которое все необходимые действия, связанные с преобразованием файлов производит непосредственно на мобильном устройстве. Хочется надеяться, что необходимая для этого адаптация алгоритма преобразования графического файла из 2D в 3D для использования под управлением операционной системы Android (после создания реализации этого алгоритма по Ubuntu, основная проблема – возможность использования OpenCV под соответствующую платформу – должна решаться аналогичным образом) не доставит много хлопот.

## 5.2. Разработка мобильного приложения

Рассмотрю практический пример, когда программно запускаю приложение «Камера», а полученную фотографию сохраняю в папке. [6]

В манифесте нужно добавить разрешение на запись файла в хранилище и указать требование наличия камеры.

Используем статическую константу ACTION\_IMAGE\_CAPTURE из объекта MediaStore для создания намерения, которое потом нужно передать методу startActivityForResult(). Разместим на форме кнопку и ImageView, в который будем помещать полученный снимок. Полученное с камеры изображение можно обработать в методе onActivityResult()

При тестировании примера на своём телефоне я обнаружил небольшую проблему - когда снимок передавался обратно на моё приложение, то оно находилось в альбомном режиме, а потом возвращалось в портретный режим. При этом полученный снимок терялся. Поэтому перед нажатием кнопки я поворачивал телефон в альбомный режим, чтобы пример работал корректно. Поэтому надо предусмотреть подобное поведение, например, запретить приложению реагировать на поворот и таким образом избежать перезапуска Activity.

По умолчанию фотография возвращается в виде объекта Bitmap, содержащего миниатюру. Этот объект находится в параметре data, передаваемом в метод onActivityResult(). Чтобы получить миниатюру в виде объекта Bitmap, нужно вызвать метод getParcelableExtra() из намерения, передав ему строковое значение data.

(рисунок 20)

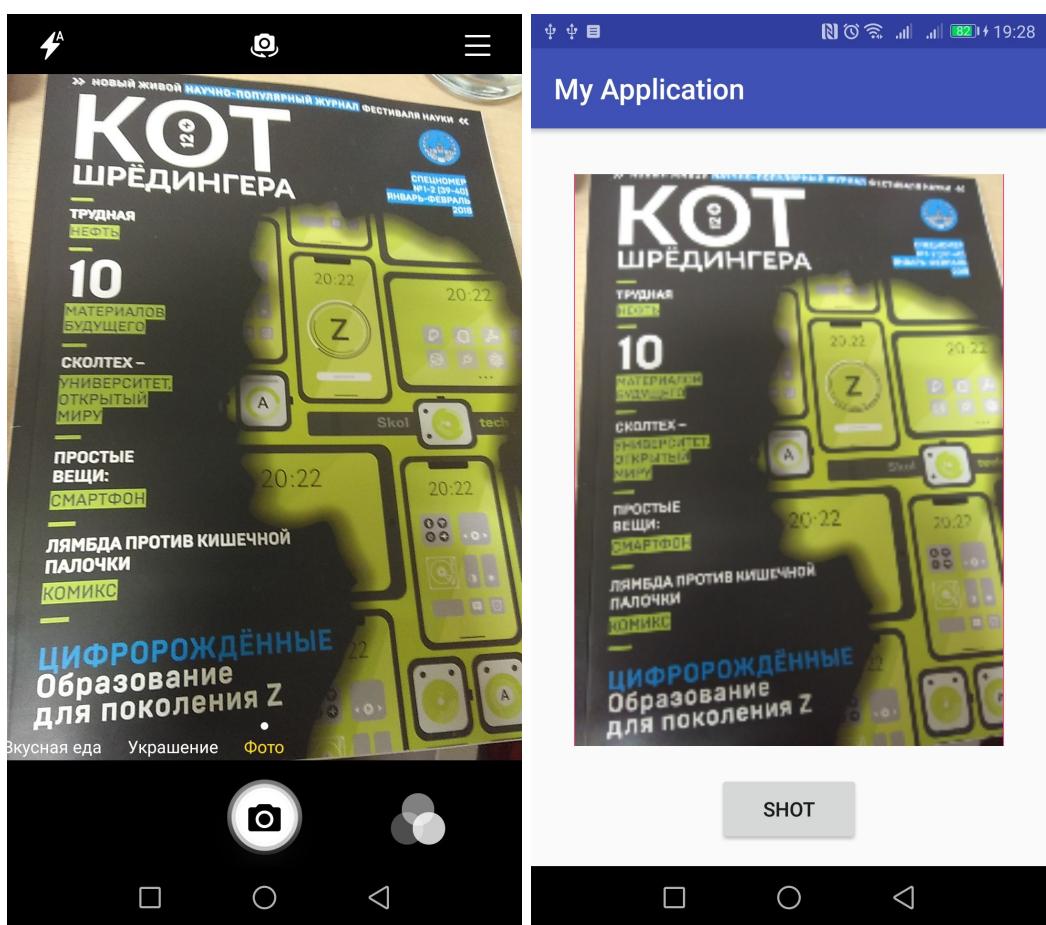


Рисунок 20 – Результат работы приложения

## ЗАКЛЮЧЕНИЕ

В ходе проделанной работы изучил основы разработки мобильного приложения в среде Android Studio. Также были исследованы методы «дефокусировки» и семантического анализа, была изучена работа алгоритма определения глубины изображения.

При создании пользовательского интерфейса приложения были проанализированы современные, с аналогичным функционалом мобильные приложения в целом. Был сделан акцент на необходимости создания современного, функционального и не перегруженного пользовательского интерфейса. В результате проведенного анализа был создан пользовательский интерфейс мобильного приложения для преобразования 2D фотографий в 3D вид.

Рассмотрев различные варианты создания мобильного приложения, было решено создать веб-сервис, который предоставляет минимально необходимый пользовательский интерфейс для загрузки желаемого файла на сервер, преобразования файла на сервере и, как результат, возможность скачать получившийся файл на устройство пользователя и поделиться этим файлом в социальных сетях. Алгоритм преобразования графического файла из 2D в 3D работает. В ближайшей перспективе реализовать его в виде серверного модуля.

## СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. *Soetaert K., Petzoldt T., Setzer R. W.* Solving differential equations in R // The R Journal. — 2010. — Dec. — Vol. 2, no. 2. — P. 5–15. — URL: [http://journal.r-project.org/archive/2010-2/RJournal\\_2010-2\\_Soetaert~et~al.pdf](http://journal.r-project.org/archive/2010-2/RJournal_2010-2_Soetaert~et~al.pdf).
2. ВЫЧИСЛЕНИЕ КАРТЫ ГЛУБИНЫ СТЕРЕОИЗОБРАЖЕНИЯ НА ГРАФИЧЕСКОМ ПРОЦЕССОРЕ В РЕАЛЬНОМ ВРЕМЕНИ. — 2012. — URL: <https://www.fundamental-research.ru/ru/article/view?id=30010>.
3. *Iddan G.J. Y. G.* 3D imaging in the studio and elsewhere. — Proc. SPIE., 1994. — C. 48—55.
4. *Fehn C. Cooke E. S. O.* 3D analysis and imagebased rendering for immersive TV applications. — 2002. — C. 705—715.
5. Material Design. —. — URL: <https://material.io/>.
6. Taking Photos. —. — URL: <https://developer.android.com/training/camera/photobasics.html#TaskPhotoView>.
7. *V. P. Namboodiri S. C.* Recovery of relative depth from a single observation using an uncalibrated (real-aperture) camera. — CVPR, 2008.
8. *Hecht E.* Optics (4th Edition). — Addison Wesley, 2001.
9. *Canny J.* A computational approach to edge detection. — IEEE Trans., 1986. — C. 679—698.
10. *G. Petschnigg R. S.* Digital photography with flash and no-flash image pairs. — ACM Trans, 2004. — C. 664—672.
11. *A. Levin D. Lischinski Y.* Colorization using optimization. — ACM Trans, 2004. — C. 689—694.
12. *A. Levin D. Lischinski Y. W.* closed-form solution to natural image matting. — IEEE Trans., 2008. — C. 228—242.

## Приложение А

Листинг 1 – Часть кода реализации класса `MainActivity` (создание мобильного приложения)

```

29             } catch (IOException ex) {
30                     // Error occurred while creati
31             }
32         if (photoFile != null) {
33             Uri photoURI = FileProvider.get
34             Intent mediaScanIntent = new I
35             File f = new File(mCurrentPhot
36             mediaScanIntent.setData(photoU
37             this.sendBroadcast(mediaScanIn
38             pictureIntent.putExtra(MediaSt
39             photoURI);
40             startActivityForResult(picture
41             1);
42         }
43     }
44 }
45
46 private File createImageFile() throws IOException {
47     // Create an image file name
48     String timeStamp = new SimpleDateFormat("yyyyM
49     String imageFileName = "JPEG_" + timeStamp + ".J
50     File storageDir = getExternalFilesDir(Environment.
51     File image = File.createTempFile(
52         imageFileName, /* prefix */
53         ".jpg", /* suffix */
54         storageDir /* directory */
55     );
56
57     // Save a file: path for use with ACTION_VIEW
58     mCurrentPhotoPath = image.getAbsolutePath();
59     return image;
60 }
61

```

```
62         @Override  
63         protected void onActivityResult(int requestCode, int r  
64             if (requestCode == 1 && resultCode == RESULT_O  
65                 Bitmap imageBitmap = (Bitmap) data.get  
66                 ivPreview.setImageBitmap(imageBitmap);  
67             }  
68         }  
69     }
```

---

## Приложение В

Листинг 2 – Фрагмент кода вычисления значений градиента

---

```
1  
2 #include "_3DAnaglyph.h"  
3 float gh[9] = {  
4     1,0,-1,  
5     2,0,-2,  
6     1,0,-1  
7 };  
8 float gv[9] = {  
9     -1,-2,-1,  
10    0,0,0,  
11    1,2,1  
12};  
13 void MakeMaskGradient(IplImage* src1, IplImage* mask, IplImage* gr)  
14 {  
15     unsigned char* b = (unsigned char*)src1->imageData;  
16     unsigned char* m = (unsigned char*)mask->imageData;  
17     float* grad = (float*)gr->imageData;  
18     int x,y, xx, yy;
```

```

19     cvZero(gr);
20
21     for (y = 1; y < src1->height-1; y++)
22     for (x = 1; x < src1->width-1; x++)
23     {
24
25         if (m[y*src1->widthStep + x] > 0)
26         {
27
28             float nx = 0;
29
30             float ny = 0;
31
32             int k = 0;
33
34             for (yy = y - 1; yy <= y + 1; yy++)
35             {
36
37                 for (xx = x - 1; xx <= x+1; xx++,k++)
38
39                     nx += b[yy*src1->widthStep + xx];
40
41                     ny += b[yy*src1->widthStep + xx];
42
43             }
44
45             grad[y*src1->width + x] = sqrt(nx*nx+ny*ny);
46
47         }
48
49         else
50
51             grad[y*src1->width + x] = 0;
52
53     }
54
55 }
```

---