МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«КРЫМСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ имени В. И. ВЕРНАДСКОГО»
(ФГАОУ ВО «КФУ им. В. И. Вернадского»)
Таврическая академия (структурное подразделение)
Факультет математики и информатики
Кафедра информатики

Иванча Николай Александрович

Разработка элементов дополненной реальности для визуализации туристических объектов Крыма

Выпускная квалификационная работа

Обучающегося 4 курса

Направления подготовки 01.03.02. Прикладная математика и информатика

Форма обучения очная

Научный руководитель
Доцент кафедры информатики,
Кандидат физико-математических наук М. Г. Козлова

К ЗАЩИТЕ ДОПУСКАЮ:

Заведующий кафедрой,
доктор физико-математических наук,
профессор В. И. Донской

Симферополь, 2020

Аннотация

Иванча Н. А. Разработка объектов дополненной реальности для визуализации туристических объектов Крыма. Выпускная квалификационная работа бакалавра по направлению 01.03.02. Прикладная математика и информатика. Таврическая академия Крымского федерального университета имени В. И. Вернадского.

Рассмотрены этапы создания мобильного приложения дополненной реальности с распознаванием текста. Содержание работы состоит в исследовании, тестировании и разборе современных технологий, позволяющих работать с дополненной реальностью и распознаванием текста. Результатом работы является создание мобильного приложения.

Ключевые слова: дополненная реальность, AR, распознавание, текст, оптические системы распознавания, OCR, бинаризация, маркер, объект, модель, моделирование, программное обеспечение, среда разработки, предобработка, изображение, методы, классификация, сегментация, обучение, анализ, приложение, визуализация, город, Крым.

Страниц – 50, таблиц – 0, иллюстраций – 21, приложений – 8, библиографических источников – 32.

Annotation

Ivancha N. A. Development of augmented reality objects for visualization of tourist objects in Crimea. Final qualification work of the bachelor in the direction 01.03.02. Applied mathematics and computer science. Taurida Academy of the V. I. Vernadsky Crimean Federal University.

The stages of creating an augmented reality mobile app with text recognition are considered. The content of the work consists of research, testing and analysis of modern technologies that allow working with augmented reality and text recognition. The result is the creation of a mobile app.

Keywords: augmented reality, AR, recognition, text, optical recognition systems, OCR, binarization, marker, object, model, simulation, software, development environment, preprocessing, image, methods, classification, segmentation, training, analysis, application, visualization, city, Crimea.

Pages-50, tables-0, illustrations-21, appendices-8, bibliographic sources-32.

Содержание

Введение

Изучая рынок современных технологий, можно сделать вывод, что технология дополненной реальности (AR) востребована во многих сферах жизни человека. Такая актуальность обосновывается следующими причинами:

1) возможность получать информацию в реальном времени;
2) необычный способ представления информации, благодаря которому возможно привлечь больше внимания окружающих и более точно донести до них необходимую информацию;
3) усиливает запоминание деталей и понимание процесса за счёт наглядности;
4) актуально для сферы образования;
5) инновационность: несмотря на уверенное положение на рынке технологий, воспринимается как нечто новое и современное;
6) большой, постоянно расширяющийся спектр сфер использования.

Учитывая высокую популярность AR технологий, отличным решением будет внедрение элементов дополненной реальности в самую востребованную отрасль для Крыма – туризм. С помощью дополненной реальности можно визуализировать достопримечательности крымского полуострова на физической карте, с целью привлечения внимания большего числа туристов с разных уголков земного шара.

Целью работы является разработка программного обеспечения реализующего визуализацию туристических объектов Крыма в дополненной реальности.

Задачи, решаемые в работе:

1) обзор научных источников по данному вопросу;
2) изучение необходимых программных средств и библиотек;
3) создание 3D-моделей достопримечательностей;
4) реализация программного продукта для распознавания названий городов на изображении (карте);
5) реализация программного продукта для визуализации 3D-моделей достопримечательностей.

1. Распознавание текста

Задача распознавания текста особенно актуальна в наше время. Существует огромное количество печатных изданий, играющих важнейшие роли в развитии человечества, которые находятся далеко не в лучшем состоянии. И несмотря на то, что многим хотелось бы сохранить их первозданный вид, они всё же нуждаются в оцифровке во избежание потери информации. Для решения этой задачи придётся применить технологии распознавание текста.

Распознавание текста – это один из важнейших разделов распознавания образов с широчайшей сферой применения. Во-первых, это, упомянутая ранее, оцифровка. Помимо старинных печатных изданий, которые отцифровывают для сохранения культурного наследия, данная технология имеет большой спрос и для работы с современной печатной продукцией.

Программы распознавания текста невероятно востребованы среди представителей современного бизнеса. Главным образом они служат для автоматизации ввода и обработки данных из документов, за счёт чего помогают экономить время и деньги. Текст отсканированного документа, его фотографию или PDF-файл можно просматривать с экрана компьютера, но их содержимое нельзя копировать и изменять. Благодаря технологии распознавания текста исходное изображение может быть переведено в формат, доступный для редактирования. В качестве примера программного обеспечения, реализующего данную технологию, можно привести систему оптического распознавания текста ABBYY OCR, которая занимает лидирующие позиции в бизнес сфере [27].

Технология распознавания текста также используется для перевода текста с картинки с одного языка на другой. Основная область применения таких приложений – сфера туризма. Распознавание текста в данном случае используется для мгновенного перевода географических наименований, которые могут встретиться туристу в незнакомой для него стране. Примером такого использования технологии распознавания текста является приложение для мобильных устройств Google Переводчик. Данное приложение поддерживает мгновенный перевод любых надписей с 38

языков на русский, а также поддерживает распознавание рукописного ввода с 93 языков, что тоже является результатом работы программной части, отвечающей за распознавание текста [7].

Следует отметить, что процесс распознавания текста может быть реализован при помощи четырёх основных методов:

1) сравнение с подготовленными шаблонами;

2) распознавание, которое использует критерии распознаваемого объекта;

3) распознавание, основанное на структуре объекта;

4) распознавание при помощи нейронных сетей, самообучающихся алгоритмов.

1.1. Оптическое распознавание символов

Оптическое распознавание символов (optical character recognition или OCR) – это электронное или механическое преобразование изображений машинописного, рукописного или печатного текста в машинно-кодированный текст, будь то отсканированный документ, фотография документа, фотография сцены (например, текст на вывесках и рекламных щитах в альбомной фотографии) или из текста субтитров, наложенного на изображение (например, из телевизионной трансляции).

Процесс оптического распознавания символов можно разделить на следующие этапы:

1) Предобработка

Оптические системы распознавания текста часто предварительно обрабатывают изображения, чтобы повысить шансы на успешное распознавание. К таким методам можно отнести следующие методы: выравнивание (de-skew) – если документ не был выровнен должным образом при сканировании, может возникнуть необходимость наклонить его на несколько градусов по часовой стрелке или против часовой стрелки, чтобы сделать линии текста идеально горизонтальными или вертикальными; удаление шумов (despeckle) – удаление положительных и отрицательных пятен, сглаживание краев; бинаризация (binarisation) – преобразование изображения из цветного в черно-белое с целью отделения текста (или любого другого желаемого компонента изображения) от фона;

удаление линий (line removal) – удаление нерельефных частей изображения; анализ макета или «зонирование» – определение столбцов, абзацев, подписей и прочих элементов как отдельные блоки; обнаружение линий и слов (line and word detection) – устанавливает базовую линию для слов и символов, при необходимости разделяет слова; изоляция символов или «сегментация» (character isolation or «segmentation») – разделение изображения на сегменты с одним или несколькими символами на каждом, распознавание символов и дальнейшее соединение их; нормализация соотношения сторон и масштаба (normalize aspect ratio and scale).

2) Распознавание текста

Существует два основных типа алгоритма распознавания текста. Матричное сопоставление включает в себя сравнение изображения с сохраненным шаблоном, сравнивая пиксели; оно также известно, как «сопоставление шаблонов», «распознавание образов» или «корреляция изображений». Это зависит от того, насколько входной символ правильно изолирован от остальной части изображения, а сохранённый шаблон имеет такой же шрифт и масштаб. Этот метод лучше всего работает с машинописным текстом и плохо работает, когда встречаются новые шрифты.

Извлечение объектов разлагает символы на «объекты», такие как линии, замкнутые контуры и «перекрёстки». Выделение признаков уменьшает размерность представления и делает процесс распознавания вычислительно эффективным. Эти характеристики сравниваются с абстрактным векторным представлением символа, которое может быть сведено к одному или нескольким прототипам шаблона. Общие методы обнаружения признаков в компьютерном зрении применимы к данному типу распознавания, который обычно наблюдается в «интеллектуальном» распознавании почерка и в большинстве других современных программах распознавания.

Такие программы, как Cuneiform и Tesseract, используют двухпроходный подход к распознаванию символов. Второй проход известен как «адаптивное распознавание» и использует формы букв, распознанные с высокой степенью уверенности на первом проходе, чтобы лучше распознавать оставшиеся буквы на втором проходе. Это выгодно для

необычных шрифтов или низкокачественных сканов, где шрифт искажён (например, размытый или выцветший).

Современное программное обеспечение для распознавания символов, например OCRopus или Tesseract, использует нейронные сети, которые были обучены распознавать целые строки текста вместо того, чтобы фокусироваться на отдельных символах.

3) Постобработка

Точность распознавания может быть увеличена, если выходные данные ограничены лексиконом – списком слов, которые могут встречаться в документе. Это могут быть, например, все слова в русском языке или более технический лексикон для конкретной области. Этот приём может быть проблематичным, если документ содержит слова, отсутствующие в лексиконе, например имена собственные. Tesseract использует свой словарь, чтобы повлиять на шаг сегментации символов, для повышения точности. Выходной поток может представлять собой обычный текстовый поток или файл символов, но более сложные системы распознавания текста могут сохранять исходный макет страницы и создавать, например, аннотированный PDF-файл, включающий как исходное изображение страницы, так и текстовое представление с возможностью поиска.

«Анализ ближайшего соседа» может использовать частоты встречаемости для исправления ошибок, отмечая, что некоторые слова часто встречаются вместе. Например, «Вашингтон, округ Колумбия» обычно гораздо чаще встречается в английском языке, чем «Вашингтон DOC». Знание грамматики сканируемого языка также может помочь определить, является ли слово глаголом или существительным, например, что позволяет повысить точность.

4) Оптимизация для конкретных приложений

Всё больше и больше поставщиков технологий распознавания начали настраивать системы распознавания таким образом, чтобы более эффективно работать с конкретными типами входных данных. Помимо специфичного для конкретного приложения лексикона, более высокая производительность может быть достигнута за счёт учёта бизнес-правил, стандартных выражений или богатой информации, содержащейся в цветных изображениях. Эта стратегия называется «Ориентированность на

приложение распознавания» или «Индивидуально-настраиваемая система распознавания» и применяется для распознавания номерных знаков, счетов, скриншотов, удостоверений личности, водительских удостоверений и производства автомобилей.

The New York Times адаптировала технологию OCR в собственный инструмент, который они называют Document Helper, что позволяет их интерактивной команде новостей ускорить обработку документов, которые необходимо просмотреть. Они отмечают, что это позволяет им обрабатывать до 5400 страниц в час, обрабатывая материал так, чтобы репортёры могли ознакомиться с его кратким содержанием [11].

1.2. Факторы влияющие на точность распознавания

На точность распознавания символов влияет большое количество факторов, многие из которых могут быть устранены в процессе предобработки. Следующие факторы оказывают наибольшее влияние на результат распознавания:

1) Лингвистическое разнообразие. Многие языки обладают одинаковым буквами: «O», «A», «P» и др. или похожими: «N» и «И», «R» и «Я» и др. В каждом языке также существуют буквы похожие между собой: «M», «H», «И» – в русском, «I», «L», «J» – в английском. Некоторые буквы имеют сходства с цифрами «O» и «0», «I» и «1» и т. д.

2) Искажение изображения. К возможным искажениям изображения можно отнести описанные ранее: нарушение перспективы, соотношения сторон и т. д.

3) Защита источника изображения. Часто встречается на документах с персональными данными. В случае паспортов – это невидимые символы.

Разнообразные вариации размеров, шрифтов и масштабов символов. Каждый отдельный символ может быть написан с использованием различных шрифтов, как стандартных (Times, Orator, Arial), а также множеством шрифтов, используемых, например, в пределах одной страны, для оформления одних и тех же документов.

4) Эффекты освещения (тени, блики и т. п.) при съёмке камерой. Многие системы распознавания символов показывают наилучшие результаты при обработке чёрного текста на светлом (в идеале – белом)

фоне. В случае текстовых документов – это не критично. Но если применять такую систему для обработки нестандартных изображений: сканы паспортов, водительских удостоверений, номерных знаков автомобилей, то мы столкнёмся с невозможностью получить корректные данные без предварительной обработки изображения [25].

1.3. Методы распознавания символов

Технология распознавание различных печатных символов с изображений находит своё применение среди множества научных и прикладных задач, которые нацелены на идентификацию всевозможных объектов в реальном мире. К числу задач, решаемых с помощью OCR, можно отнести не только класс задач по распознаванию текста, но и ряд других специализированных задач, ориентированных на распознавание символической информации, нанесённой на поверхность различных объектов. В зависимости от класса задачи применяют один из следующих методов распознавания символов.

1.3.1. Шаблонные методы

Шаблонные или растровые методы распознавания текста основаны на сравнении исходного изображения с заранее подготовленными шаблонами, используя некую метрику. Метрика – некоторое условное значение функции, определяющее положение объекта в пространстве. В случае, если два объекта расположены близко друг к другу, то метрики для таких объектов будут совпадать или быть предельно похожими, это будет означать, что два объекта представляют, например, одну и ту же букву, написанную разными шрифтами. В качестве метрики можно выбрать расстояние Хэмминга.

Данную метрику часто используют при кодировании информации и передаче данных, она показывает, как сильно объекты не похожи между собой. Например, после сеанса передачи на выходе имеется следующая последовательность бит (1000001), также нам известно, что должна прийти другая последовательность бит (1000101). Мы вычисляем метрику путём сравнения частей последовательности с соответствующими местами из другой последовательности. Таким образом метрика Хэмминга в нашем

случае равна 3. Так как объекты отличаются в трёх позициях. 3 – это степень непохожести, чем больше, тем хуже в нашем случае.

Следовательно, чтобы определить какая буква изображена нужно найти её метрику со всеми готовыми шаблонами. И тот шаблон, чья метрика окажется наименьшей будет ответом.

Но реализовав систему распознавания используя расстоянии Хэмминга можно прийти к выводу, что подсчёт одной лишь метрики не даёт положительного результата, так как многие буквы похожи между собой. Например «j» и «i» в английском алфавите, их метрика близка к нулю, что не гарантирует однозначности распознавания и приводит к ошибочному результату.

Решением данной проблемы является отличная от расстояния Хэмминга метрика, которая позволяет разграничить некоторое множество букв в отдельный класс. Примером такой метрики является метрика, основанная на горизонтальном и вертикальном отражении объектов и преобладании одного из отображений.

Такие буквы как «H» «I» «i» «O» «o» «X» «x» «l» обладают суперсимметрией (полностью совпадают со своими отражениями и значимые пиксели распределены равномерно по всему изображению), они выносятся в отдельный класс, что сокращает перебор всех метрик примерно в 6 раз. При проведении аналогичных действий в отношении других букв можно достигнуть уменьшения перебора примерно в 3 раза. Особой уникальностью обладает буква «J», которая находится в своём классе одна, и, следовательно, идентифицируются однозначно.

Следующем шагом может быть использована метрика Хэмминга, которая на данном этапе даёт лучшие показатели, чем при прямом применении [20].

Реализовав систему распознавания текста на основе данного метода мы пришли к тому, что из-за огромного количества различных печатных изданий и используемых в них разнообразных модификаций шрифтов рассмотреть все случаи на этапе обучения невозможно.

К достоинствам данного метода можно отнести: простоту программной реализации, стабильную работу в условиях отсутствия

помех, а также высокую точность распознавания символов, имеющих различные дефекты.

Но также данный метод имеет существенные недостатки, в числе которых: сильная «шрифтозависимость», сильная зависимость от качества предварительно подобранных шаблонов, медленная работа при большом количестве помех, а также резкое ухудшение точности распознавания при работе с повёрнутыми, наклонными и другими изображениями, имеющими различные искажения.

1.3.2. Признаковые методы

Признаковые методы являются наиболее распространёнными [19], признак в них определяется как функция от значений, содержащихся в одном или более пикселях, и вычисляется так, что численно выражает некоторую значимую характеристику объекта.

Все признаки присущие символам можно классифицировать следующим образом:

– признаки пиксельного уровня: признаки, вычисляемые в каждом пикселе, такие как цвет, положение;

– локальные признаки: признаки, вычисляемые в некотором окне или ограниченной области изображения;

– глобальные признаки: признаки, вычисляемые по всему изображению. Обычно, это статистические свойства изображений, например, гистограмма, среднее значение, дисперсия и другие статистические моменты.

С другой стороны, все признаки могут быть условно разделены на низкоуровневые и высокоуровневые признаки. Низкоуровневые признаки могут быть извлечены непосредственно из исходного изображения, тогда как высокоуровневые признаки базируются на низкоуровневых признаках.

Множество подлежащих обработке символов разбивается на конечное число классов, называемых образами. Объекты каждого класса (образы) характеризуются некоторой совокупностью признаков, каждый из которых принимает одно из определённого множества значений. Пространство признаков разбивается на непересекающиеся области, каждая из которых соответствует определённому классу символов. Для

предъявленного к распознаванию изображения символа вычисляются все признаки, значения которых задают положение некоторой точки в пространстве признаков. После этого определяют, в какую из областей пространства признаков попала данная точка, и исходный символ относят к классу символов, связанному с данной областью [30, 29].

Исходя из этого, к достоинствам данного метода можно отнести высокую обобщающую способность, а также устойчивость к изменению формы символов, что позволит распознавать с помощью данного метода различные шрифты и начертания.

В качестве недостатков можно выделить неустойчивость к различным дефектам изображения и потерю информации о символе на этапе извлечения признаков

1.3.3. Структурные методы

Структурные методы переводят символ в его топологическое представление, отражающее информацию о взаимном расположении структурных элементов. В качестве основных структурных представлений изображений используются детекторы линий, полос, пятен или углов, находящихся в соединениях отрезков вида «Г», «Т», «Y» и «X» [31, 29].

Также для построения описания изображений символов используется контурное представление. Построение структурных элементов на основе контуров может выполняться путём сегментации контуров, их аппроксимации или обнаружения точек максимальной кривизны. К контурным структурным элементам относятся: отрезки прямых линий, дуги окружностей или эллипсы, углы и т. д. [32, 29].

При проведении структурного сравнения изображений производится поиск соответствий между структурными элементами с учётом их взаимного расположения, типа, размеров, взаимной ориентации и т.д. Таким образом, в структурных методах вместо поиска в множестве параметров пространственного преобразования производится поиск в множестве вариантов соответствий структурных элементов [29].

Среди достоинств данного метода можно выделить способность распознавания искажённых символов и хорошие показатели точности при работе с различными типами шрифтов.

К недостаткам же стоит отнести неустойчивость к различным дефектам изображения, таким как «слипание» символов или «разрыв» структурных элементов одного символа [25].

1.3.4. Нейросетевые методы

Нейросетевые методы основаны на применении различных типов искусственных нейронных сетей. Идея этих методов – моделирование работы мозга человека. Нейрон, как структурная единица нейронной сети, получает множество входных сигналов. В нашем случае входные сигналы описывают значение пикселя изображения, то есть, если имеется изображение 16x16, входных сигналов у нейрона должно быть 256. Каждый входной сигнал воспринимается с определенным коэффициентом и в результате, по окончанию распознавания на каждом нейроне скапливается определённый заряд, у какого нейрона заряд будет больше тот и испустит импульс. А так как все нейроны поименованы значениями букв, следовательно, среагировавший нейрон и несёт ответ распознавания.

Обучение нейронных сетей происходит на множестве обучающих примеров. Из вариантов обучения можно использовать обучение с учителем (персептрон) или самоорганизацию (сеть Кохонена). Процесс обучения можно описать следующим образом. Берётся очередное изображение из обучающей выборки и отправляется для обработки в построенную нейронную сеть. Сеть анализирует все позиции чёрных пикселей (цвет символов) и выравнивает коэффициенты минимизируя ошибку совпадения методом градиента, после чего определённому нейрону сопоставляется данное изображение.

Основными преимуществами данного метода являются способность к обобщению и высокая скорость работы.

Но также этому методу присущи и существенные недостатки: чувствительность к вращению и искажению символов, высокая зависимость от обучающей выборки и алгоритма обучения [25, 20].

1.4. Методы бинаризации изображений

Большинство OCR использует бинаризацию изображений (выделение текста из фона) как первый шаг в процессе распознавания.

Выполнив бинаризацию изображения на этапе предобработки и передав результат к готовому модулю распознавания текста, можно значительно повысить точность относительно прямого распознавания.

Процесс бинаризации представляет собой перевод цветного изображения или изображения в градациях серого в двухцветное – черно-белое. Основной параметр данного преобразования – порог, со значением которого затем сравнивается яркость всех пикселей. После сравнения, пикселю присваивается одно из двух возможных значений: 0 – «граница объекта» или 1 – «остальная область» [18].

Общая схема бинаризации представлена на рисунке Рис. 1.1.

Рис. 1.1. Общая схема бинаризации изображения.

В зависимости от порога методы бинаризации можно условно разделить на две группы: первая группа использует фиксированный порог для заданного изображения (глобальная бинаризация), тогда как вторая группа использует локальные пороги (локальная бинаризация).

1) Глобальная бинаризация.

Глобальная бинаризация хорошо применима в случаях, когда текст занимает большую часть изображения и хорошо контрастирует с фоном. При реализации такого метода находится порог бинаризации t, с помощью которого происходит деление на чёрное и белое, причём величина порога t остаётся неизменной в течение всего процесса бинаризации. К глобальным методам бинаризации относятся:

- бинаризация с нижним порогом;
- бинаризации с верхним порогом;
- бинаризация с двойным ограничением;
- неполная пороговая обработка;
- многоуровневое пороговое преобразование [10].

Метод бинаризации изображения с нижним порогом является одним из самых простых, так как в нём рассматривается только одно значение порога. Пусть P – это пиксель исходного изображения, тогда $F(P)$ – яркость пикселя P, а $\Phi(P)$ – функция переводящая пиксель в бинарное представление. В случае для *бинаризации с нижним порогом* справедлива следующая формула:

$$\Phi(P) = \begin{cases} 0, F(P) \geq t, \\ 1, F(P) < t. \end{cases} \quad (1.1)$$

Если в приведённой выше формуле для точки изображения выполняется первое условие, то такая точка является точкой объекта, если же выполняется второе условие, то точка будет точкой фона.

Метод *бинаризации с верхним порогом* можно считать частным случаем бинаризации с нижним порогом, при котором получается негатив исходного изображения. Это достигается при помощи следующей формулы:

$$\Phi(P) = \begin{cases} 0, F(P) \leq t, \\ 1, F(P) > t. \end{cases} \quad (1.2)$$

Если необходимо выделить определённые области, значения яркости пикселей в которых могут изменяться в некотором диапазоне, то применяется метод *бинаризации с двойным ограничением*, который является объединением двух предыдущих методов и выражается формулой:

$$\Phi(P) = \begin{cases} 0, F(P) \geq t_1, \\ 1, t_1 < F(P) \leq t_2, \\ 0, F(P) > t_2. \end{cases} \quad (1.3)$$

Чтобы получить наиболее простое для дальнейшего распознавание изображение, на котором будет отсутствовать фон со всеми его деталями, следует воспользоваться методом *неполной пороговой обработки*, используя формулу:

$$\Phi(P) = \begin{cases} 0, F(P) \leq t, \\ F(P), F(P) > t. \end{cases} \quad (1.4)$$

В случае, если необходимо получить изображение, которое содержит в себе n сегментов, обладающих различной яркостью, можно применить метод *многоуровневого порогового преобразования*. Однако, при этом,

полученное в ходе преобразования изображение уже не будет являться бинарным. Формула данного преобразования представлена ниже:

$$\Phi(P)=\begin{cases}1, F(P)\in T_1,\\ 2, F(P)\in T_2\\ \quad\dots\\ n, F(P)\in T_n,\\ 0, \textit{иначе.}\end{cases} \tag{1.5}$$

Одним из самых популярных методов бинаризации является метод Оцу [18]. С помощью данного метода вычисляется порог t, минимизирующий среднюю ошибку сегментации, т. е. среднюю ошибку от принятия решения о принадлежности пикселей изображения объекту или фону. В таком случае значения яркостей пикселей изображения можно рассматривать как случайные величины, а их гистограмму – как оценку плотности распределения вероятностей. Если плотности распределения вероятностей известны, то можно определить оптимальный порог для сегментации, разбивающий изображения на два класса: объекты и фон.

К достоинствам метода Оцу можно отнести:

– простота реализации;

– адаптация к различного рода изображениям, при помощи выбора оптимального порога;

– быстрое время выполнения [10].

2) Локальная (адаптивная) бинаризация.

С другой стороны, методы локальной бинаризации могут лучше справляться с неравномерным освещением и изменением цвета текста, но они более чувствительны к выбору параметров.

Самым известным представителем данного метода бинаризации является алгоритм Ниблэка [9]. При его применении для каждого пикселя *P* с координатами *(x, y)* локальный порог устанавливается по формуле:

$$\Phi(x,y)=\mu(x,y)+k\sigma(x,y), \tag{1.6}$$

где μ – математическое ожидание, σ – дисперсия, а k – параметр, определяющий какую именно часть границы объекта необходимо взять в качестве объекта.

Хотя в большинстве случаев алгоритм Ниблэка дает результат, хорошо подходящий для задач распознавания (не теряет контрастные детали), данный алгоритм плохо справляется со слабовыраженными структурами в области фона – может появляться «грязь» (Рис. 1.2.).

Данный недостаток послужил причиной появления ряда идейно близких алгоритмов, иногда именуемых ниблэковскими: Саволы, Вольфа, Фенга, алгоритм NICK. Обобщая, можно сказать, что они вводят дополнительные параметры, позволяющие устранить указанный дефект [22].

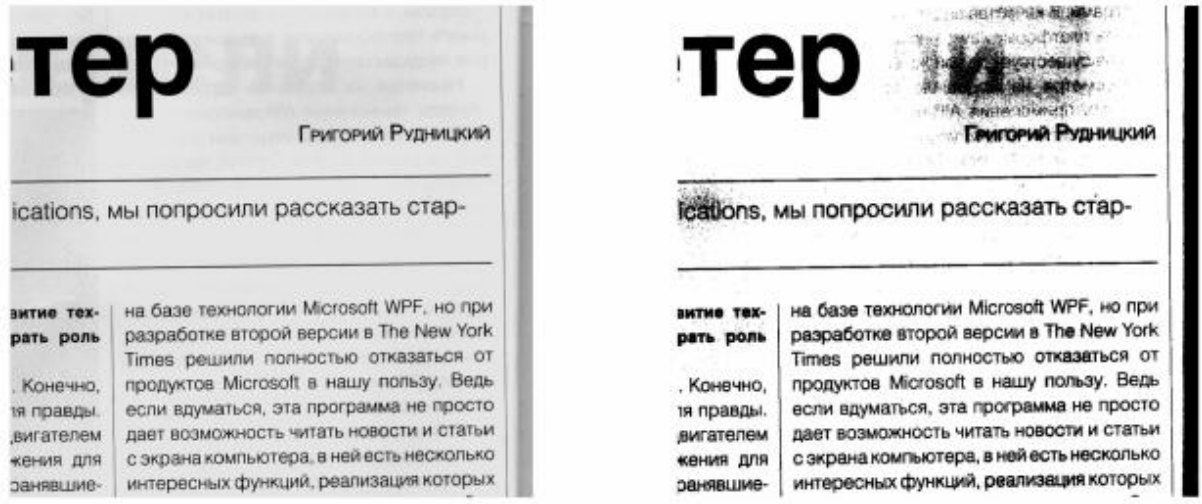


Рис. 1.2. Изображение и результат его бинаризации (k = 0.2).

Выбор метода бинаризации должен зависеть от конкретной задачи, однородности фона, качества изображения и т. д.

1.5. Коммерческие OCR инструменты

Существует множество программ оптического распознавания символов, большинство из которых являются коммерческими продуктами. К числу самых успешных из них можно отнести:

1) Google Drive OCR или Google Cloud Vision

В число доступных инструментов входят:

– Label Detection – детектирование класса к которому относится изображение, детектирование того, что изображено на изображении: котики, собачки, слоники, безмятежность, и т. д.

– OCR — распознавание текста.

– Explicit Content Detection – детектирование запрещённого контента.

– Facial Detection – детектирование лиц, черт лиц, особых точек на лицах.

– Landmark Detection – детектирование геолокации по фотографии.

– Logo Detection – детектирование символов и знаков.

Google поддерживает множество API, связанных с искусственным интелектом, например, обработку естественного языка, распознавание голоса, глубокое обучение и зрение.

Cloud Vision API [2] даёт возможность интегрировать в своё приложение модуль, который будет классифицировать контент по категориям. Также он обеспечивает разработчика информацией о достоверности, т. е. насколько вероятно, что то, что он считает отнёс к некоторому классу объектов, действительно присутствует на изображении. Google Cloud Vision API отлично подходит для интеграции в приложения дополненной реальности, в особенности, для приложений созданных чтобы помочь людям с ограниченными возможностями. Также уникальная функция Landmark Detection может быть успешно интегрирована в приложения, помогающие туристам ориентироваться на незнакомой местности. Для распознавания текста Google Cloud Vision API не является лучшим выбором, так как показатели точности не являются максимальными и отсутствует возможность точной настройки и управления процессом распознавания.

2) ABBYY FineReader

ABBYY FineReader [27] – это система оптического распознавания текстов, которая предназначена для конвертирования в редактируемые форматы отсканированных документов, PDF-документов и файлов изображений документов, включая цифровые фотографии.

Последняя, вышедшая на данный момент, версия этой OCR – ABBYY FineReader 15 предоставляет пользователь набор следующих инструментов:

– сканирование и конвертирование бумажных и PDF-документов в редактируемые форматы с возможностью полнотекстового поиска и дальнейшего редактирования;

– сравнение версий документов, даже если они созданы в разных форматах;

– создание снимков любой части экрана и последующее распознавание текста и сохранение в редактируемом формате.

Как стационарная программа для распознавания текста и конвертирования изображений в редактируемые форматы, ABBYY FineReader значительно опережает всех своих конкурентов, в том числе благодаря удобному пользовательскому интерфейсу. Но отсутствие бесплатной версии и возможности интеграции в собственные программные продукты является огромным минусом данного варианта.

3) CuneiForm

Cognitive OpenOCR [6] – свободно распространяемая открытая система оптического распознавания текстов российской компании Cognitive Technologies. Она может захватывать материал с подключенных сканеров, а также импортировать информацию из всех распространенных форматов изображений. Приложение способно автоматически определять границы для текста, который следует обработать, после распознавания позволяет внести правки в печатные данные с помощью любого текстового редактора.

CuneiForm является менее мощным бесплатным аналогом ABBYY FineReader, но также как и своего более мощного конкурента, её невозможно интегрировать в собственное приложение.

1.6. Tesseract – движок OCR с открытым исходным кодом

Tesseract – движок OCR с открытым исходным кодом, завоевавший популярность среди разработчиков OCR. Обретать свою популярность он начал сразу после разработки компанией HP между 1984 и 1994 годами. С 2006 года разрабатывается компанией Google.

Tesseract можно использовать для извлечения печатного текста из изображений напрямую или с помощью API. Он поддерживает широкий спектр языков, но не имеет встроенного графического интерфейса. Tesseract совместим со многими языками программирования и средами разработки через, написанные неравнодушными сторонними разработчиками, пользовательские интерфейсы.

Алгоритм работы Tesseract довольно прост, его можно представить следующей схемой (Рис. 1.3.).

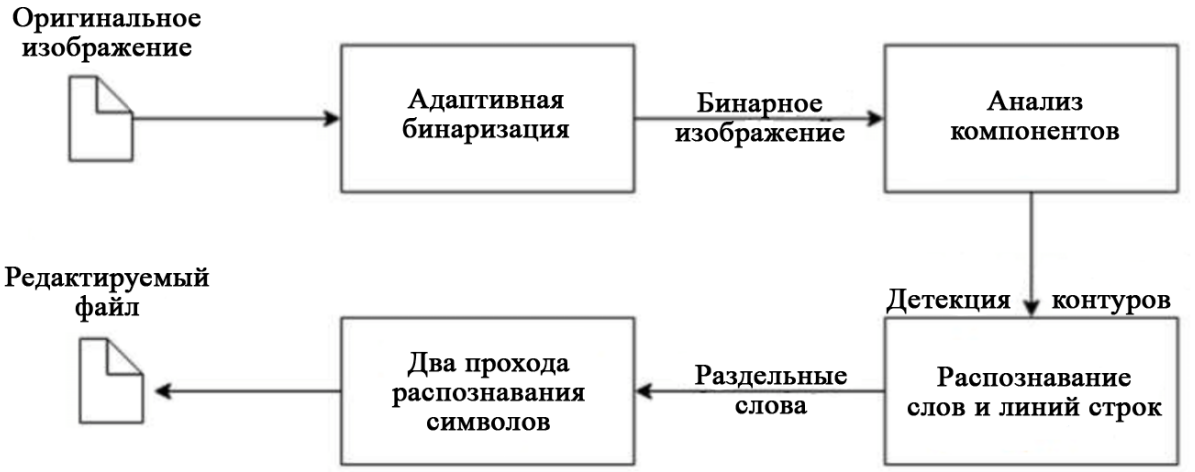


Рис. 1.3. Алгоритм работы Tesseract OCR.

Этапы распознавания текста с помощью Tesseract:

1) Поиск строк.

Алгоритм поиска строк является одной из основных частей Tesseract OCR, он разработан таким образом, что перекошенное изображение может быть распознано без предварительного выравнивания, что позволяет избежать потери качества изображения. Ключевыми частями этого процесса являются фильтрация объектов изображения и построение линий. Предполагая, что анализ макета страницы уже предоставил текстовые области примерно одинакового размера, простой фильтр высоты удаляет выпадающие элементы и вертикально соприкасающиеся символы. Средняя высота приближается к размеру текста во всём слове, поэтому можно безопасно отсекать все те объекты, которые меньше средней высоты (знаки препинания, диакритические знаки и шум).

Отфильтрованные объекты с большей вероятностью будут соответствовать модели непересекающихся, параллельных, но при этом наклонных линий. Сортировка и обработка точек по координате X предоставляет возможность определения их к уникальной текстовой строке. При этом отслеживание наклона по всей странице, значительно снижает опасность присвоения объекта к неправильной текстовой строке. После того, как отфильтрованные точки были присвоены линиям, для установки базовых линий, формирующих строки, используется наименьшее среднее квадратов. Отфильтрованные контуры помещаются обратно в соответствующие строки.

На заключительном этапе процесса создания строки происходит слияние контуров, которые перекрываются по крайней мере наполовину по горизонтали, а также нанесение диакритических знаков вместе с правильной базовой линией и связывание частей поломанных символов.

2) Выбор базовой линии.

После того, как строки текста были найдены, ограничительные линии устанавливаются более точно с помощью квадратичного сплайна. Это помогает лучше обрабатывать изображения со скошенными строками, которые могут появляться, например, при сканировании книг с переплётом.

Базовые линии устанавливаются путем разделения всех точек на группы относительно исходной базовой линии. Квадратичный сплайн устанавливается в области с наибольшей плотностью распределения (предполагается что это базовая линия). Квадратичный сплайн имеет преимущество в стабильности, но недостаток в том, что, при использовании нескольких сегментов сплайна, могут возникать разрывы. Более традиционный кубический сплайн работает в данном случае лучше.

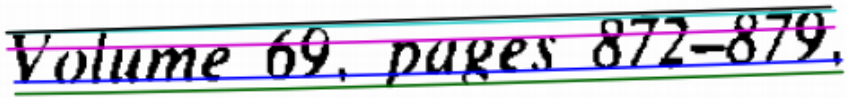


Рис. 1.4. Базовые линии

На Рис. 1.4 показан пример строки с текстом и подогнанной к нему базовой, нижней, средней и верхней линиями. Все эти линии параллельны между собой и слегка изогнуты. Над верхней линией (голубая, печатается как светло-серая) находится чёрная линия, которая является ориентиром (прямой). Относительно неё голубая (светло-серая) линия немного изогнута.

3) Определение фиксированного шага и разделение слова на буквы.

Tesseract проверяет текстовые строки, чтобы определить, как в них распределены буквы. Если буквы расположены с фиксированным шагом, то Tesseract разбивает это слово на символы (Рис. 1.5.).

Рис. 1.5. Разбиение слова с фиксированным шагом

4) Поиск слов.

В случае с нефиксированным шагом или пропорциональным интервалом могут возникать проблемы, изображённые на Рис. 1.6.

Расстояние между десятками и единицами в «11,9%» такое же, как и расстояние между остальными буквами, и, безусловно, больше, чем расстояние между словами «Federated» и «junk». Tesseract решает проблемы такого рода путем измерения зазоров между базовой линией и средней линией. Области, которые в результате этой проверки оказались близки к порогу, трудно распознаваемы, поэтому окончательное решение об их распознавании может быть принято после распознавания слов.

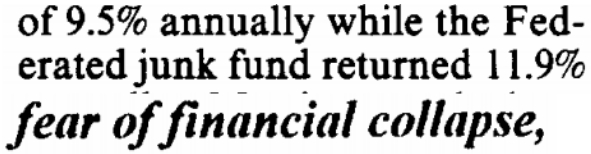


Рис. 1.6. Текст с нефиксированным шагом

5) Распознавание слов.

Частью процесса распознавания для любого OCR является определение того, как слово должно быть сегментировано на символы. В первую очередь классифицируется исходная сегментация, полученная в результате линейного поиска. Остальная часть распознавания на этом этапе применяется только к тексту без фиксированного шага.

6) Разделение «слипшихся» символов.

Если результат распознавания неудовлетворителен, Tesseract пытается улучшить его, вырезая символ, в котором он меньше всего уверен, из классификатора символов. Кандидатуры точек разделения находятся из вогнутых вершин многоугольного контура и могут быть либо другой вогнутой вершиной, либо сегментом линии. Для успешного отделения соединенных символов из набора «ASCII» может потребоваться до 3 пар таких точек. Пример разделяющих точек представлен на Рис. 1.7.

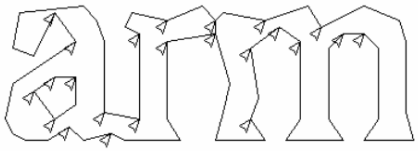


Рис. 1.7. Возможные точки разделения символов

7) Распознавание дефектных символов.

Когда потенциальные варианты разбиения были исчерпаны, а слово все еще не распознано, оно передаётся ассоциатору. Данный модуль совершает поиск сегментированных символов в возможной комбинации максимально обрезанных контуров, чтобы найти символ максимально похожий на обрубленный элемент.

Можно утверждать, что этот подход в лучшем случае неэффективен, а в худшем случае очень вероятна потеря важных сегментов разделения. Преимущество заключается в том, что эта схема упрощает структуры данных, которые потребуются для поддержки полного графа сегментации.

Таким образом, слова с разорванными символами, такие как на Рис. 1.8, могут быть успешно распознаны классификатором Tessaract OCR.

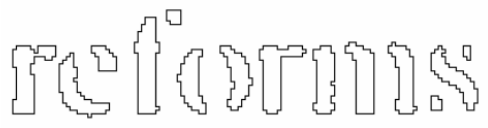


Рис. 1.8. Слово с дефектными буквами

8) Классификация.

Классификация протекает как двухэтапный процесс. На первом этапе классификатор создает короткий список классов, к которым может принадлежать неизвестный символ. Каждый признак извлекается из битового вектора классов, которым он может соответствовать, и битовых векторов, просуммированных по всем признакам. Классы с наибольшим количеством значением (после корректировки ожидаемого количества символов) становятся новым списком для следующего шага.

Для каждого признака неизвестного символа ищется битовый вектор прототипов данного класса, которому он может соответствовать, а затем вычисляется фактическое сходство между ними. Каждый класс символов прототипа представлен логическим выражением суммы произведения с каждым термином, называемым конфигурацией, поэтому процесс вычисления расстояния между ними сохраняет общее сходство каждого признака в каждой конфигурации, а также каждого прототипа. Лучшее комбинированное расстояние, которое вычисляется из суммированных признаков и свидетельств прототипа, является лучшим из всех сохраненных конфигураций класса.

9) Обучающая выборка.

Несмотря на то, что классификатор способен легко распознавать поврежденные символы, он не был обучен на них. Для обучения данного классификатора было задействовано 20 выборок из 94 символов восьми шрифтов одного размера с четырьмя начертаниями (обычный, полужирный, курсив, жирный курсив), что составляет в общей сложности 60160 обучающих символов. Это существенно отличается от других опубликованных классификаторов, таких как классификатор Калера с

более чем миллионом выборок и 100-шрифтовый классификатор Бэрда с 1175000 обучающими символами.

10) Лингвистический анализ.

Каждый раз, когда модуль распознавания слов рассматривает новую сегментацию, лингвистический модуль выбирает наилучшее доступное слово в каждой из следующих категорий: часто встречающееся слово, часто встречающееся слово из словаря, часто встречающаяся цифра, часто встречающееся слово верхнего регистра, часто встречающееся слово нижнего регистра, предполагаемое классификатором слово. Окончательное решение для данной сегментации – это просто слово с наименьшей суммарной оценкой расстояния, где каждая из вышеперечисленных категорий умножается на некоторую константу.

Слова, которые были по-разному сегментированы, скорее всего, содержат разное количество символов. Сравнивать такие слова напрямую довольно сложно. Несмотря на то, что классификатор сможет посчитать вероятности их совпадения, Tesseract не сможет достоверно определить их сходство [13].

1.7. Глубокое обучение, как метод распознавания

Глубокое обучение – это метод машинного обучения. Глубокое обучение позволяет обучать модель предсказывать результат по набору входных данных. Для обучения сети можно использовать как контролируемое, так и неконтролируемое обучение. Отличие машинного обучения от глубокого представлено на Рис. 1.9 [16].

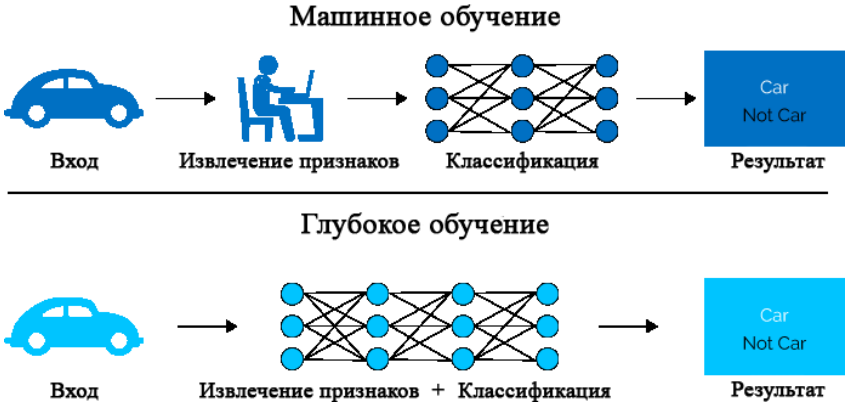


Рис. 1.9. Отличие машинного обучения от глубокого

Нейронная сеть глубоко обучения состоит из нескольких слоёв, на каждом из которых располагаются взаимосвязанные нейроны. Существует три вида слоёв: входной, скрытый (скрытые) и выходной. Слово «глубина»

в термине «глубокое обучение» означает наличие более чем одного скрытого слоя. Скрытые слои выполняют математические вычисления со входными данными. После того, как набор входных данных прошел через все слои нейронной сети, функция активации возвращает выходные результаты через выходной уровень.

Для обучения нейронной сети можно использовать открытую нейросетевую библиотеку Keras. Она нацелена на оперативную работу с сетями глубинного обучения, при этом спроектирована так, чтобы быть компактной, модульной и расширяемой.

Обучение любой модели в машинном обучении начинается с данных. Keras содержит внутри несколько обучающих датасетов, но в их числе, к сожалению, отсутствует набор кириллических символов. Поэтому в качестве обучающего датасета можно воспользоваться CoMNIST (Cyrillic-oriented MNIST).

Датасет CoMNIST включает в себя 118 символов заглавные и строчные латинские, кириллические буквы, а также цифры. Чтобы не возникало неоднозначного определения букв (из-за схожести многих латинских символов с кириллическими) латинские буквы не будут участвовать в обучении. В таком случае остаётся 66 символов, а следовательно, 66 выходов у обучаемой нами нейронной сети. Так как символы в датасете CoMNIST хранятся как картинки размером 28×28 пикселей, то на вход нейросеть будет получать изображение с аналогичным размером. Для этого необходимо на этапе предобработки определить слово на исходном изображении и разбить его на буквы, которые в дальнейшем привести к формату 28×28 пикселей.

Для обучения нейронной сети нам необходимо разделить выборку на две части: тренировочные данные и тестирующие. Брать весь датасет или какую-то его часть зависит от вычислительных мощностей компьютера и от времени, предполагаемого затратить на процесс обучения. Но также от размера выборки зависит точность распознавания обученной модели.

2. Дополненная реальность

В последние годы несложно заметить стремительное развитие технологий, цель которых облегчить деятельность человека. Одной из таких технологий является augmented reality (AR) или дополненная реальность. Дополненная реальность – это технология, которая в реальном времени дополняет привычный для человека физический мир, элементами компьютерной графики (картинки, текст и др.) и аудиоэффектами, используя при этом специальное программное обеспечение и доступные каждому – смартфоны, планшеты или другие подобные устройства.

Очень важно отличать дополненную реальность от виртуальной (virtual reality, VR) и смешанной (mixed reality, MR). В отличие от дополненной реальности, где виртуальные объекты проецируются на реальное окружение, виртуальная представляет собой новый, созданный техническими средствами мир, который передаётся человеку через органы зрения, слуха и осязания. Смешанная реальность является наиболее сложной, так как объединяет оба подхода.

Дополненная реальность уверенно занимает лидирующие позиции на рынке технологий. Это подтверждает столь частое упоминание AR в презентациях тех или иных IT-гигантов. Такой интерес к дополненной реальности вызван тем, что она предоставляет возможность взаимодействовать с окружением на принципиально новом уровне, а также имеет огромный спектр областей применения.

Основным двигателем AR технологий на данный момент является индустрия развлечений, которая видит в дополненной реальности возможность установления более тесных связей между аудиторией и своими персонажами. Так, например, в конце 2016 года компанией Niantic была выпущена мобильная игра про «карманных монстров» Pokemon GO, которая всего за несколько месяцев побила рекорды популярности среди мобильных приложений и положила начало кардинальным изменениям в индустрии развлечений [21].

Не обошли стороной новые технологии и рекламную индустрию. С появлением дополненной реальности различные компании открыли для себя новые способы донесения привлекательности своего продукта до своих потенциальных потребителей. Дополненную реальность используют для наполнения плоских поверхностей листовок и билбордов интерактивным контентом, который имеет огромное преимущество перед

статичной рекламой. Например, известный журнал об автомобилях Top Gear предоставил своим читателям возможность просматривать видеоролики об автомобилях прям на страницах издания, используя для этого только лишь смартфон со специальным приложением. Такой же рекламный ход совершила и компания Faberlic, предоставив своим клиентам возможность посмотреть на косметические товары из своего печатного каталога в натуральную величину, прочитать состав на этикетке и сделать заказ [17].

Несмотря на популярность дополненной реальности в сферах рекламы и развлечения, потенциал использования AR намного шире, что наглядно подтверждают современные примеры применения технологии в медицине. Первый самый очевидный пример использования AR в медицине – это организация максимально наглядного обучения студентов медицинских вузов. Наглядную анатомию в дополненной реальности способно продемонстрировать приложение HoloAnatomy, разработанное компанией Microsoft для очков смешанной реальности HoloLens [8].

Более серьёзными с точки зрения медицины являются приложения: MITK pille, которое позволяет, используя планшет, прямо во время операции «заглянуть» в тело пациента, и приложение для людей с ограниченными возможностями зрения и слуха Aira, в котором реализован нейросетевой помощник, распознающий и проговаривающий всё, что видит камера очков [14].

Своё применение дополненная реальность нашла и в сфере дизайна и проектирования. Так, например, компания Ikea интегрировала AR в своё приложение Ikea Place, с помощью которого позволяет пользователям проверить, как мебель может выглядеть в их домах [12].

В условиях карантина дополненная реальность особенно актуальна для следующих услуг:

– примерка одежды, обуви, часов и других аксессуаров при использовании интернет-магазинов;

– примерка косметики, окрашивание волос;

– инструкции к самостоятельному ремонту бытовой техники;

– обучение в дополненной реальности.

Также дополненная реальность оказалась востребована в сфере туризма. Уже существуют приложения, помогающие людям, оказавшимся в незнакомом городе, успешно ориентироваться на местности и прокладывать маршруты к необходимым локациям. Примером такого

приложения является разработка компании MapWay – приложение Bus Times London, которое, используя камеру смартфона, помогает туристам найти свою автобусную остановку в пределах города Лондон, рисуя средствами дополненной реальности путь до неё [4].

При посещении музеев, памятников архитектуры или других достопримечательностей средствами дополненной реальности могут быть реализованы такие опции как виртуальный экскурсовод или визуализация каких-либо событий, связанных с данной достопримечательностью.

Основа технологии дополненной реальности – это система оптического трекинга. Камера распознает маркеры в реальном мире, «переносит» их в виртуальную среду, накладывает один слой реальности на другой и таким образом создает мир дополненной реальности.

Существуют три основных направления в развитии этой технологии:

1) «Безмаркерная» технология AR.

Основой «безмаркерной» зарядки является алгоритм распознавания, при котором на окружающий ландшафт, снятый камерой, накладывается виртуальная «сетка». На этой сетке программные алгоритмы находят некие опорные точки, по которым определяют точное место, к которому будет «привязана» виртуальная модель. Такая технология имеет преимущество в том, что в роли маркеров выступают объекты реального мира и для них не нужно создавать специальных визуальных идентификаторов, что делает эту технологию доступной вне зависимости от геолокации её использования.

2) AR технология на базе маркеров.

Технология на базе специальных маркеров, или меток, удобна тем, что они проще распознаются камерой и дают ей более жесткую привязку к месту для виртуальной модели. Такая технология гораздо надежнее «безмаркерной» и работает практически без сбоев. Но недостатком такой технологии является необходимость предварительной разработки и внедрения маркеров в продукты подлежащие взаимодействию с приложением AR.

3) «Пространственная» технология.

Кроме «маркерной» и «безмаркерной», существует технология дополненной реальности, основанная на пространственном расположении объекта. В ней используются данные GPS/ГЛОНАСС, гироскопа и компаса, встроенного в мобильный телефон. Место виртуального объекта определяется координатами в пространстве. Активация программы

дополненной реальности происходит при совпадении координаты, заложенной в программе, с координатами пользователя. Самым очевидным минусом такой технологии является погрешность в распознавании координат пользователя, что может привести к неправильному отображению виртуальных объектов [28].

2.1. Готовые инструменты AR

Существует ряд готовых инструментов для работы с виртуальной или дополненной реальностью. В большинстве случаев использование таких платформ облегчает работу с элементами дополненной реальности, исключая из разработки собственного продукта работу со сложными геометрическими преобразованиями и визуализацию. Самыми популярными из таких платформ являются: Wikitude, ARToolKit, Vuforia.

2.1.1. Wikitude

Данный фреймворк предоставлен онлайн-студией для наложения простых статичных объектов дополненной реальности [23]. Алгоритм работы довольно прост: загрузить целевое изображение в студию, добавить объекты AR, сгенерировать JavaScript код и вставить в свой проект. Таким образом весь рендеринг осуществляется средствами ArchitectView от Wikitude JS SDK. Такой подход позволяет легко интегрировать дополненную реальность в свой проект, учитывая также, что данный фреймворк поддерживает работу с Unity и интеграцию C++ плагинов. Дерево возможных интеграций Wikitude представлено на Рис. 2.1.

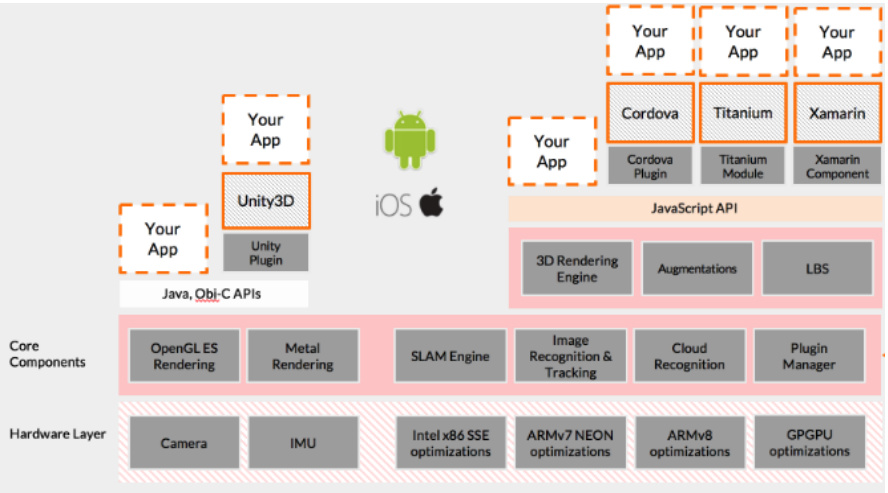


Рис. 2.1. Дерево возможных интеграций Wikitude

Wikitude SDK является мощным инструментом для создания элементов дополненной реальности, но данный инструмент является коммерческим продуктом, распространяемым по подписке.

2.1.2. ARToolKit

ARToolKit – это набор программных библиотек, которые могут быть использованы в приложениях дополненной реальности. Большим преимуществом данного инструмента является открытый исходный код. Библиотека ARToolKit позволяет разрабатывать приложения дополненной реальности для самых популярных платформ: Android, iOS, Windows и Linux. Обычно каждая операционная система требует своей собственной среды разработки для разработки приложения, но ARToolKit поддерживает самую популярную в мире платформу разработки для создания кроссплатформенных игр и интерактивного контента – Unity [26].

Основные особенности ARToolKit:

1) надежная система слежения за маркером;
2) поддержка калибровки камеры;
3) одновременное отслеживание нескольких объектов;
4) поддержка нескольких языков программирования;
5) оптимизация для мобильных устройств;
6) полная поддержка Unity3D.

Пример взаимодействия ARToolKit и Uniyu3D представлен на Рис. 2.2.

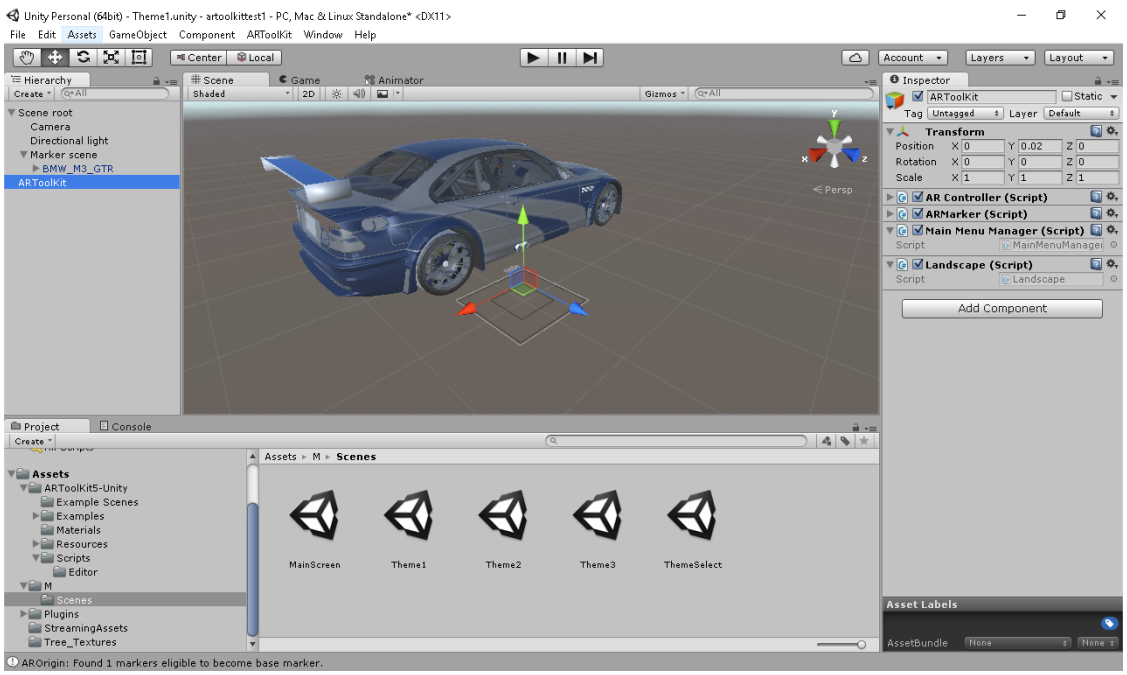


Рис. 2.2. Пример проекта AR приложения в Unity3D

Библиотека дополненной реальности ARToolKit работает по следующему принципу:

1) камера захватывает видео и отправляет кадр на обработку;

2) встроенные модули проверяют кадр на наличие квадратных маркеров;

3) если квадратный маркер найден и содержит в себе шаблон, то программа использует математические формулы для расчёта положения и ориентации маркера по отношению к камере;

4) после того как положения и ориентация камеры определены, происходит визуализация 3D модели;

5) конечный результат отображается на дисплее устройства [26].

2.1.3. Vuforia

Vuforia – это библиотека для разработки программного обеспечения дополненной реальности для мобильных устройств, которая распространяется в свободном доступе [15]. Vuforia использует технологию компьютерного зрения для распознавания и отслеживания плоских изображений и 3D-объектов в режиме реального времени. Эта возможность детекции изображений позволяет разработчикам позиционировать и ориентировать виртуальные объекты, такие как 3D-модели и другие носители информации, по отношению к объектам реального мира, когда они просматриваются через камеру мобильного устройства. Затем виртуальный объект отслеживает положение и ориентацию изображения в режиме реального времени, чтобы точка зрения зрителя на объект соответствовала точке зрения на цель. Таким образом, получается, что виртуальный объект является частью реальной сцены.

Vuforia SDK поддерживает различные типы 2D и 3D целей, включая «безмаркерные» объекты, трёхмерные объекты и VuMark (специальные Vuforia маркеры). Дополнительные функции SDK включают в себя точное определение координат устройства в пространстве, обнаружение препятствий и преград с помощью «виртуальных кнопок», а также возможность выбора цели в режиме реального времени.

Vuforia предоставляет интерфейсы прикладного программирования (API) на языках C++, Java, Objective-C++ и .NET через расширение для игрового движка Unity. Таким образом, SDK поддерживает как нативную разработку для iOS, Android и UWP, так и разработку AR-приложений в Unity, которые легко переносятся на перечисленные платформы.

2.2. Распознавание AR маркеров в реальных условиях

Нашей задачей является проекция на экран трехмерной модели фигуры, положение и ориентация которой соответствуют положению и ориентации некоторой предварительно определенной плоской поверхности. Кроме того, мы хотим сделать это в режиме реального времени, чтобы, если поверхность меняет свое положение или ориентацию, проецируемая модель делала это соответственно. Чтобы достичь этого, мы должны сначала определить плоскую поверхность, на которую предполагается делать проекцию, в кадре изображения или видео. После идентификации мы можем легко определить преобразование из эталонного изображения поверхности (2D) в целевое изображение (2D). Это преобразование называется гомографией. Также необходимо преобразовать координаты нашей 3D модели в вновь образованную систему координат и нарисовать модель на экране. Учитывая всё это, нашу задачу можно разделить на:

1) распознать опорную плоскую поверхность;
2) оценить гомографию;
3) преобразование исходной системы координат в систему координат целевой поверхности, учитывая гомографию;
4) проецирование 3D модели на изображение (в пиксельном пространстве) и ее отрисовка.

2.2.1. Распознавание целевой поверхности

Распознавание целевой поверхности будет состоять из трех основных этапов: обнаружение или извлечение признаков поверхности, описание признаков и сопоставление признаков.

1) Извлечение признаков.

Этот шаг заключается в поиске элементов изображения, которые являются признаками и эталонного изображения и целевого. Эти признаки будут впоследствии использованы для нахождения эталонного объекта в целевом изображении. Будем считать что признаки найдены, когда будет найдено определенное количество положительных совпадений между целевым и опорным изображениями. Чтобы данный метод точно находил признаки, важно иметь эталонное изображение, на котором видна только поверхность, которую можно найти в целевом изображении.

Чтобы область или точка изображения была помечена как признак, она должна иметь два важных свойства: 1) она должна представлять

некоторую уникальность, по крайней мере, локально. Хорошими примерами этого могут быть углы или края; 2) поскольку заранее не известно какими будут, например, условия ориентации, масштаба или яркости этого же объекта на изображении, где нужно распознать его, особенность в идеале должна быть инвариантной к преобразованиям, т.е. инвариантной по отношению к изменениям масштаба, вращения или яркости. Как правило, чем больше инвариант, тем лучше.

2) Описание признаков.

Как только признаки найдены, нужно найти подходящее описание информации, которую они предоставляют. Это позволит искать их на других изображениях, а также получить меру того, насколько похожи две обнаруженные особенности при сравнении. Это – то, где дескрипторы свертываются. Дескриптор обеспечивает представление информации, предоставленной признаком и его окружением. Как только дескрипторы вычислены, объект, который должен быть распознан, может быть абстрагирован к вектору признаков, который является вектором, который содержит дескрипторы ключевых точек, найденных на изображении с эталонным объектом. Существует много алгоритмов, которые извлекают признаки изображения и вычисляют их дескрипторы. К числу самых эффективных можно отнести: SIFT, SURF и ORB. Стоит иметь в виду, что форма и значения дескриптора зависят от используемого алгоритма.

3) Сопоставление признаков.

Как только найдены признаки объекта и сцены, где должен быть найден объект, и вычислены его дескрипторы, ищется совпадения между ними. Самый простой способ сделать это состоит в том, чтобы взять дескриптор каждого объекта в первом наборе, вычислить расстояние до всех дескрипторов во втором наборе и вернуть ближайший в качестве лучшего соответствия. Это подход грубой силы, существуют более сложные методы. Чтобы проверить, что найденное совпадение является наилучшим совпадением, необходимо вычислить обратное совпадение, от признака во втором наборе до признака в первом наборе. После того, как сопоставление завершено в обоих направлениях, примем в качестве действительных совпадений только те, которые выполнили предыдущее условие (являются лучшими совпадениями). Другой вариант уменьшения количества ложных срабатываний – проверить, находится ли расстояние от второго до лучшего соответствия ниже определенного порога. Если это так, то совпадение считается истинным [3].

2.2.2. Оценка гомографии

После определения опорной поверхности в текущем кадре и получения набора действительных совпадений, необходимо оценить гомографию между обоими изображениями. Необходимо найти преобразование, которое отображает точки из плоскости поверхности в плоскость изображения (рис. 2.3). Это преобразование будет обновлять каждый новый кадр, который обрабатывается.

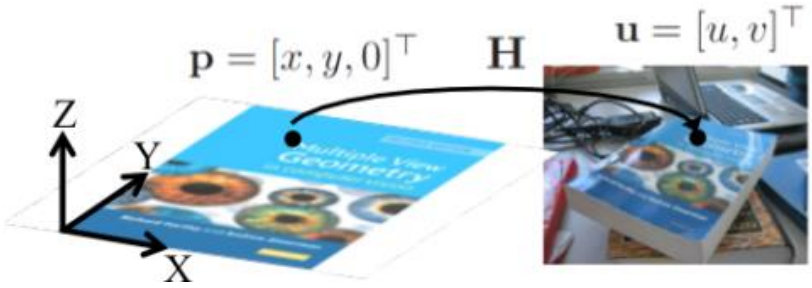


Рис. 2.3. Гомография между плоскостью и изображением [3]

Поскольку набор совпадений между обоими изображениями уже найден, осуществляется поиск однородного преобразования, которое выполняет вышеописанное отображение. Для проекции 3D модели в кадре необходимо расширить матрицу гомографии [3]. Это является важным шагом, позволяющим проектировать не только точки, содержащиеся в опорной плоскости поверхности, но и любые точки в опорном пространстве.

2.3. Разработка объектов дополненной реальности

Одним из основных этапов создания AR приложения является разработка объектов, предполагаемых к визуализации в дополненной реальности. Решением этой задачи может быть один из существующих на данный момент 3D редакторов.

3D редакторы используются для создания компьютерной графики и могут быть разделены на следующие группы:

1) программы для создания игровых сцен (игровые движки): Unreal Engine 4, Unity 5, CryEngine 3;

2) программы для скульптинга: Pixologic ZBrush, Autodesk Mudbox;

3) узкоспециализированные 3D редакторы: RealFlow – моделирование поведения жидкости, Mari – создание трёхмерных текстур;

4) универсальные 3D редакторы: Cinema 4D, 3Ds Max, Maya, Houidini, Blender и т. д.

Так как данная работа, в качестве объектов дополненной реальности, предполагает некие архитектурные сооружения (достопримечательности), то подходящим 3D редактором может стать один из универсальных. Универсальные 3D редакторы, как правило, содержат всё необходимое для 3D графики: инструменты моделирования, анимации и визуализации.

2.3.1. Cinema 4D

Cinema 4D – один из самых лучших и удобных 3D пакетов на сегодняшний день [24]. Огромный функционал: от моделирования, анимации, эффектов до «лепки» и модуля BodyPaint 3D. У Cinema 4D более понятный и удобный интерфейс нежели у 3Ds Max и Maya. Широко используется в моушен-дизайне, киноиндустрии и рекламе. К основным возможностям Cinema 4D можно отнести: генерацию и анимацию объектов, персонажную анимацию, динамику мягких и твёрдых тел, мощный графический визуализатор. Программа также позволяет просчитывать эффекты глобальной освещенности, каустику и учитывает подповерхностное рассеивание света.

Среди основных плюсов стоит отметить: лёгкость в освоении, интуитивный интерфейс, большой функционал и множество обучающих материалов. Cinema 4D распространяется как коммерческий продукт, но существует также и пробная 30-ти дневная версия. Пример работы над 3D проектом в программе Cinema 4D представлен на Рис. 2.44.

Рис. 2.4. Пример проекта Cinema 4D

2.3.2. 3ds Max

3ds Max является очень популярным инструментом во многом из-за того, что редактор от Autodesk ориентирован на архитектурную визуализацию [1]. В 3ds Max есть большое количество инструментов, необходимых при моделировании самых разных архитектурных проектов – от заготовок дверей и окон разных форм до растительности, лестниц и оград. Кроме того, в данном 3D-редакторе присутствуют средства для анализа и настройки освещенности трехмерного проекта. Также в программу был интегрирован фотореалистичный визуализатор, который дает возможность добиться высокой правдоподобности просчитываемого изображения. Пример проектирования многоэтажного здания в 3Ds Max представлен на Рис. 2.55.

Плюсами данного 3d редактора являются: большой функционал, обилие плагинов, преимущество в работе с архитектурой, а также большое количество обучающей информации.

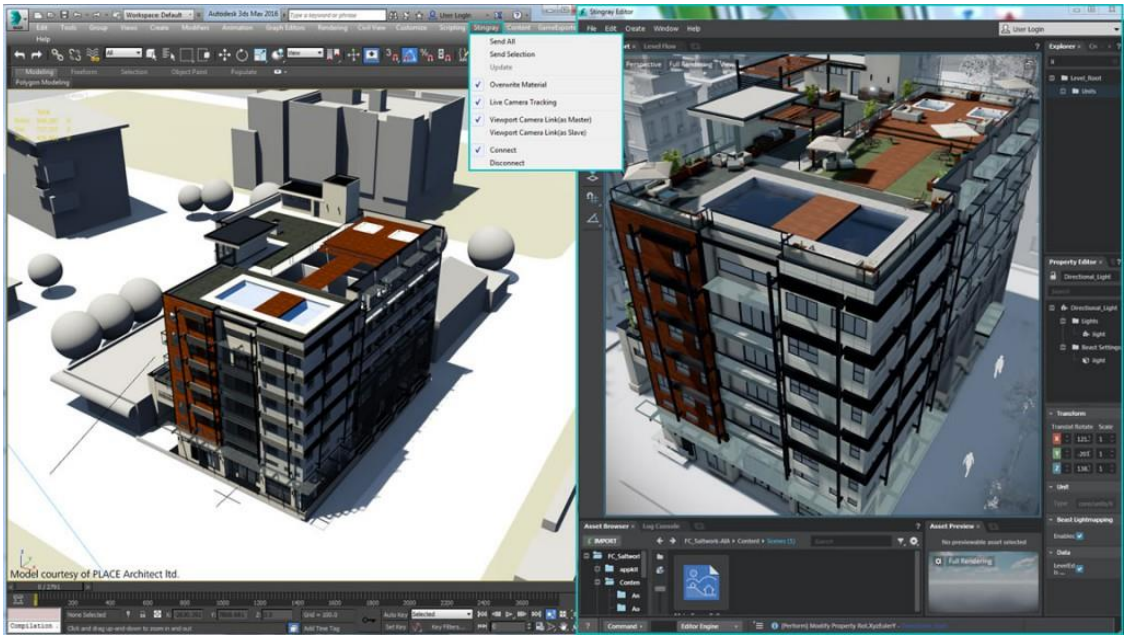


Рис. 2.5. Проектирование здания в 3Ds Max

2.3.3. Maya

Maya – промышленный стандарт 3D графики в кино и телевидении. Maya популярна среди крупных студий и масштабных проектов в рекламе, кино, игровой индустрии. Данный 3D редактор имеет огромное преимущество при создании анимации. Этот трёхмерный редактор взят на вооружение такими крупными студиями, как Pixar, WaltDisney, Dreamworks и другими. В программе есть все, что необходимо для создания трехмерной

графики. Maya позволяет пройти все этапы создания 3D – от моделирования и анимации до текстурирования, композитинга и послойного рендеринга [1]. Пример проектирования трёхмерных моделей в программе Maya представлен на Рис. 2.66.

Рис. 2.6. Проектирование в Autodesk Maya

2.3.4. Side Effects Houdini

Houdini – мощный профессиональный пакет для работы с 3D графикой, в его основе процедурная, нодовая система. Houdini идеально подходит для создания сложной динамики, симуляции: частиц, жидкости, дыма, огня, имитации природных явлений и т.д. А также это отличный

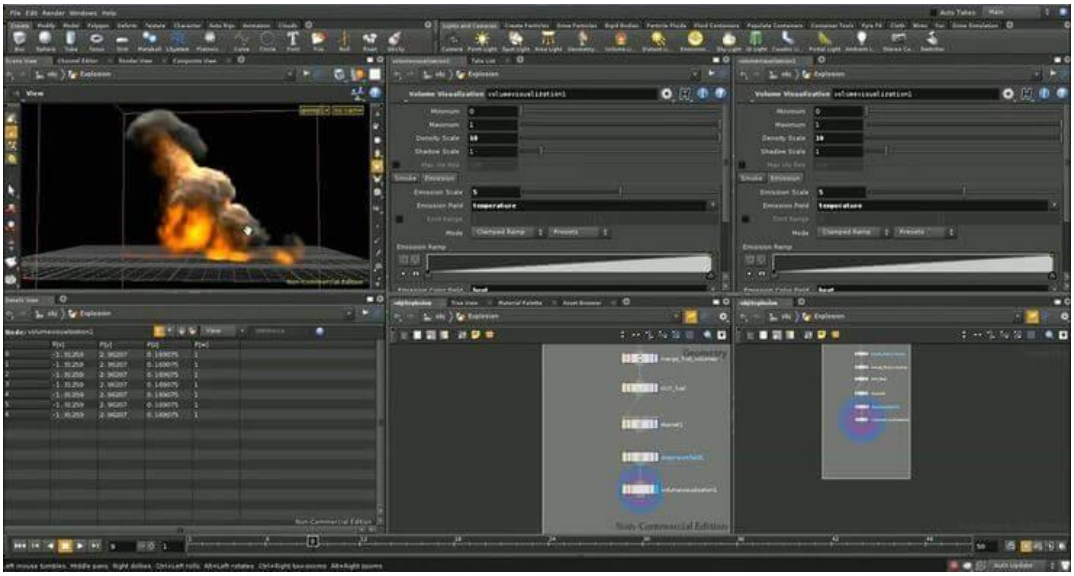


Рис. 2.7. Создание эффектов в Side Effects Houdini

инструмент для создания впечатляющих визуальных эффектов. Основная область применения Houdini – киноиндустрия.

К основным плюсам можно отнести высококлассные спецэффекты и анимацию. Но в то же время большими минусами являются: высокая цена и недостаточное количество обучающей информации [24].

На Рис. 2.77. представлен пример работы над 3D проектом в Houdini.

2.3.5. Blender

Blender является полностью бесплатным 3D редактором. Он включает в себя средства для 3D моделирования, анимации, а также набор опций для создания игр, визуальных эффектов и скульптинга. Является достойным аналогом профессиональных программ для 3D анимации. Благодаря поддержке Blender Foundation, программа очень быстро и стабильно развивается. По сравнению с коммерческими разработками размер этого редактора совершенно мизерный – всего несколько десятков мегабайт и совершено не требовательна к ресурсам компьютера. Поэтому программа может функционировать даже на ПК с очень слабыми конфигурациями, вплоть до нетбуков. Одним из самых главных плюсов программы является кроссплатформенность, что позволяет Blender одинаково хорошо работать в Linux и Windows. Помимо этого к плюсам также относятся: открытый код, широкий функционал и большое количество обучающих курсов [1]. Пример работы над 3D проектом в программе Blender представлен на Рис. 2.88.

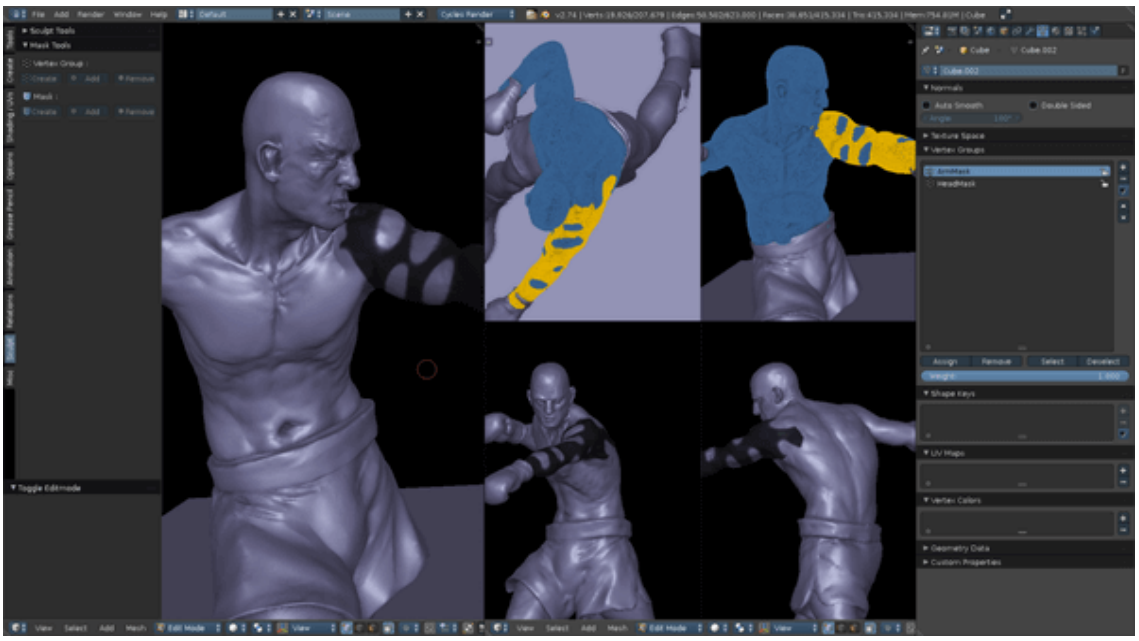


Рис. 2.8. Проектирование персонажа в Blender

3. Разработка элементов дополненной реальности для визуализации туристических объектов Крыма

3.1. Распознавание текста

Для решения задачи распознавания текста был выбран подход с использованием библиотеки Tesseract. Tesseract обладает множеством преимуществ, опираясь на которые эта оптическая система и была выбрана:

1) обладает открытым исходным кодом;
2) показывает высокие результаты на монохромных изображениях;
3) высокая скорость и простота реализации задачи распознавания текста;
4) имеет возможность работать с библиотеками компьютерного зрения (OpenCV);
5) поддерживает интеграцию в большое количество языков программирования;
6) не коммерческий продукт.

Для улучшения качества распознавания средствами OpenCV будет реализована предобработка входящего изображения, что позволит отсеять с него элементы не являющиеся частью текста.

После распознавания текста с изображения, полученная строка пройдёт фильтрацию с целью исключения не кириллических символов (знаки препинания, пробелы и т. д.). А затем, отфильтрованная строка будет сравниваться с названиями городов методом нечёткого сравнения, основанным на расстоянии Левенштейна.

В итоге мы получим строку, которая будет максимально совпадать с названием одного из городов.

3.2. Создание 3D моделей достопримечательностей

Для моделирования 3D моделей достопримечательностей Крыма был выбран 3D-редактор 3ds Max, так как он имеет большие преимущества

перед другими инструментами по части создания моделей архитектурных сооружений.

Выбор достопримечательностей был сделан на основе исторического и военно-патриотического наследия Республики Крым.

Достопримечательности были смоделированы для пяти крымских городов: Симферополь, Севастополь, Судак, Керчь, Ялта.

Город Симферополь традиционно представляет здание железнодорожного вокзала Симферополь-Пассажирский, представленного на Рис. 3.1.

Рис. 3.1. Железнодорожный вокзал Симферополь-Пассажирский.

Город-герой Севастополь представляет архитектурный символ города, установленный вблизи Приморского бульвара рядом с площадью Нахимова, – Памятник затопленным кораблям (Рис. 3.2).

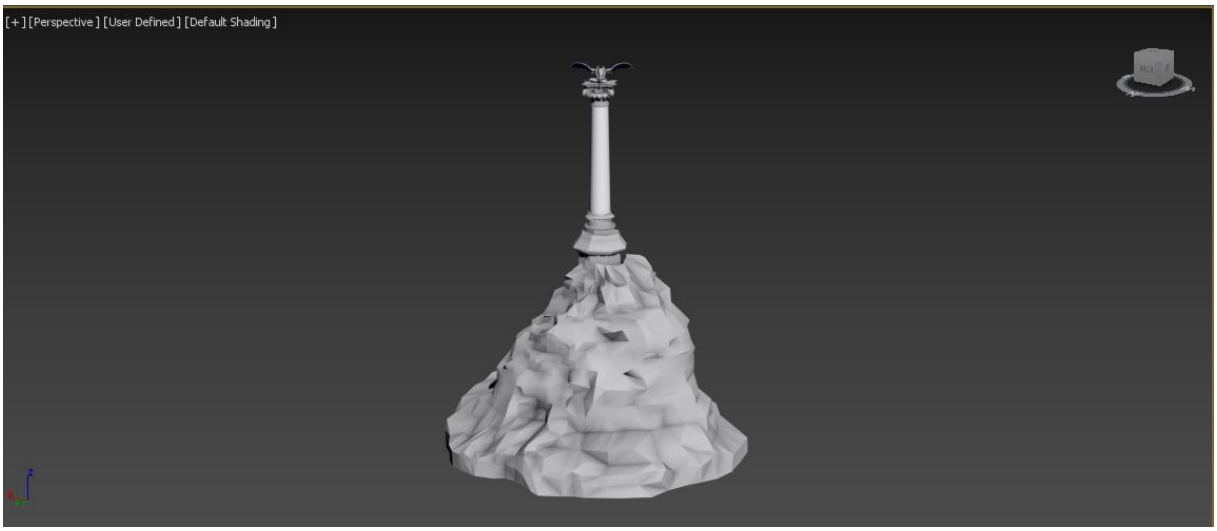


Рис. 3.2. Памятник затопленным кораблям.

Город Судак представлен фрагментом Генуэзской крепости, построенной генуэзцами как опорный пункт для своей колонии в северном Причерноморье. Выбранный фрагмент является моделью Консульской башни Генуэзской крепости, расположенной на Юго-Востоке достопримечательности (рис. 3.2.).

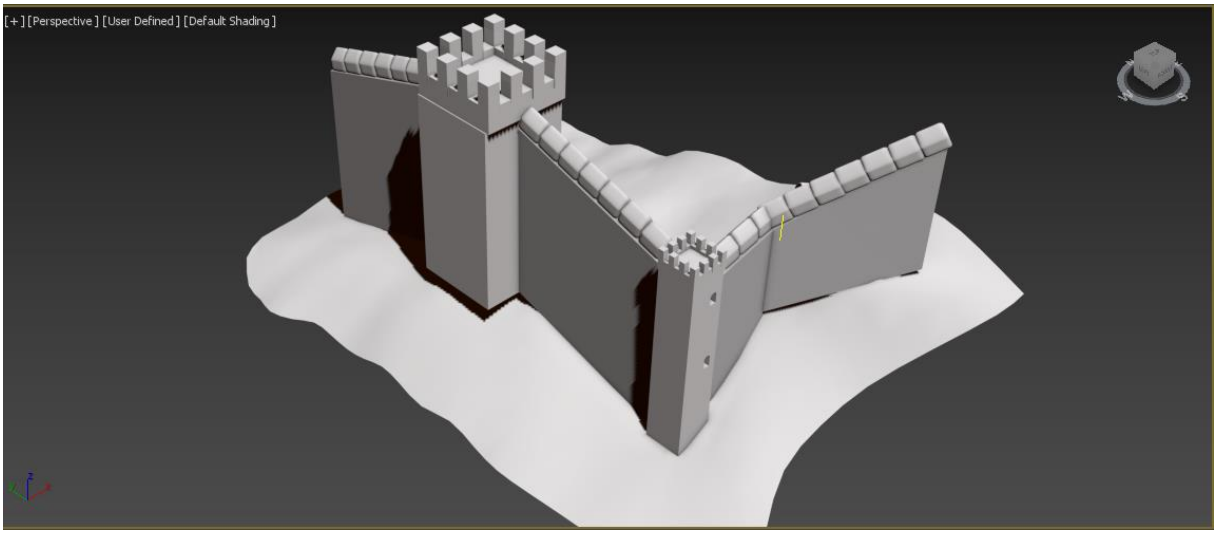


Рис. 3.3. Фрагмент Генуэзской крепости.

В качестве главной достопримечательности города-героя Керчь был выбран Обелиск Славы Бессмертным Героям на горе Митридат – монумент, посвящённый генералам, офицерам, сержантам и рядовым Отдельной Приморской Армии, морякам Азовской военной флотилии и всем воинам, павшим в боях за освобождение Крыма. Модель обелиска представлена на Рис. 3.3.

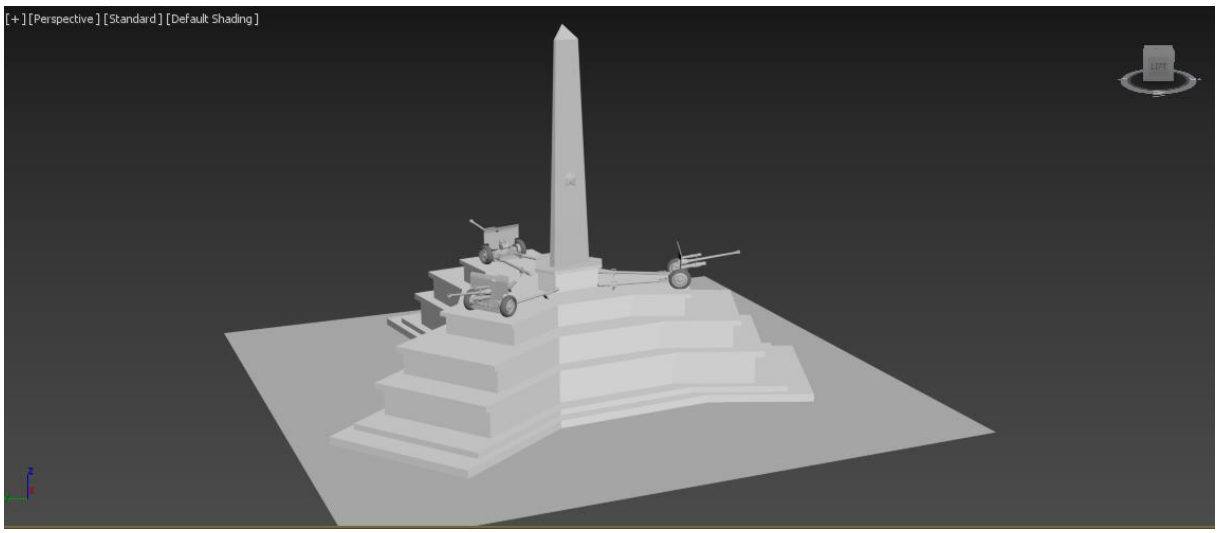


Рис. 3.3. Обелиск Славы Бессмертным Героям на горе Митридат.

Главной достопримечательностью курортной столицы Крыма – города Ялта был выбран памятник архитектуры и истории – Ласточкино

гнездо, который расположен на отвесной 40-метровой скале в посёлке Гаспра (рРис. 3.4.).

Рис. 3.4. Ласточкино гнездо

3.3. Распознавание AR маркеров

Распознавание AR маркеров и выведение 3D модели на опорную поверхность проводилось без использования готовых инструментов по следующим причинам:

1) отсутствие у представленных решений полностью бесплатных пакетов;
2) личная заинтересованность в самостоятельной реализации всех этапов визуализации 3D моделей в дополненной реальности;
3) отсутствие материально-технической базы для использования профессиональных продуктов для создания AR приложений.

3.4. Алгоритм работы приложения

Так как разрабатываемое приложение подразумевает определение ключевых слов на каждом кадре видеопотока и последующая проекция определённой (в зависимости от распознанного ключевого слова) 3D модели, то алгоритм работы реализуемого программного продукта можно вкратце описать следующим образом:

1) «Захват» кадра из видеопотока;
2) предобработка кадра для улучшения качества распознавания;
3) распознавание текста с кадра;
4) постобработка распознанного текста;

5) сравнение полученного распознавания с набором ключевых слов (названия городов).

6) в случае успешного распознавания: проекция соответствующей ключевому слову 3D модели:

6.1) определение опорной поверхности;

6.2) оценка гомографии;

6.3) преобразование координат точек 3D модели в новую систему координат;

6.4) проекция 3D модели на опорную поверхность на текущем кадре.

Если же результат распознавания не соответствует ни одному из ключевых слов, то текущий кадр остаётся без изменений.

7) Возвращение кадра в видеопоток.

Данный алгоритм реализуется на языке Python и состоит из трёх скриптов: *main.py* – реализация интерфейса и распознавание текста, *render.py* – оценка гомографии, проекция 3D модели на кадр, *objloader.py* – разбиение 3D модели на треугольники.

Все города, достопримечательности которых будут проецироваться в видеопоток, объединены в отдельный класс – *City*. При создании каждого объекта класса City, он инициализируется изображением опорной поверхности, вычисленными для неё дескрипторами и ключевыми точками, а также 3D моделью достопримечательности соответствующую определённому городу.

60 раз в секунду происходит захват кадра из видеопотока, после чего он передаётся в функцию *pasteModelIntoFrame*, которая также в качестве аргументов принимает текущий город и предыдущий успешно распознанный кадр. Данная функция проводит бинаризацию изображения и вычисляет по два ближайших дескриптора для каждой точки, отсеивает наименее похожие точки и несимметричные соответствия. После вычисляет матрицу гомографии и получает из неё и, заранее вычисленной, матрицы камеры матрицу проекции (трансформации) имеющую следующий вид:

$$\begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{pmatrix} \qquad (3.1)$$

Данная матрица трансформации включает в себя поворот *R* и перенос *T*, благодаря которым исходная модель будет спроецирована на опорную поверхность. За вычисление матрицы трансформации отвечает функция *projection_matrix*.

Проектирование модели на кадр осуществляется при помощи функции *render,* которая для каждого треугольника заданной модели проецирует все его точки на опорную поверхность и рисует полученный треугольник на сохранённой копии текущего кадра.

После отрисовки модели в кадре кадр возвращается в видеопоток. В случае недостаточного совпадения дескрипторов исходной опорной поверхности и дескрипторов кадра кадр возвращается в видеопоток без изменений.

Так как процесс распознавания текста с картинки занимает 600 миллисекунд, а кадр с видеопотока мы извлекаем 60 раз в секунду, то есть приблизительно каждые 17 миллисекунд, то отсюда следует, что распознавать текст с каждого кадра не представляется возможным. Решением этой проблемы стала поддержка языком Python многопоточности. Таким образом функция *getTextWithTesseract*, которая осуществляет предобработку полученного кадра, распознавание текста с него и последующую постобработку распознанного текста, запускается в отдельном потоке раз в секунду. Такой подход позволяет процессу распознавания работать независимо от процесса вывода кадра в видеопоток, распознавая при этом не все полученные кадры.

В зависимости от распознанного текста, в качестве текущего города назначается город соответствующий распознанию, а соответственно текущей 3D моделью становится модель достопримечательности этого самого города.

Заключение

В результате работы было разработано приложение, которое показывает хорошие результаты при различных ракурсах. Но тем не менее существует ряд направлений усовершенствования данного приложения. Так, например, в усовершенствовании нуждается блок, отвечающий за предобработку изображения, от которого непосредственно зависит точность распознавания. Также благодаря туристической ориентированности крымского полуострова список городов, чьи достопримечательности визуализируются, может быть расширен. Также для некоторых городов, которые имеют несколько характерных достопримечательностей, может быть реализована возможность переключения между последними. А разнообразие достопримечательностей Республики Крым может быть отсортировано в предусмотренные интерфейсом категории (военно-патриотические достопримечательности, религиозные и т. д.).

Система визуализации объектов дополненной реальности также может быть усовершенствована. Необходимым является добавление реалистичных текстур 3D моделей. Также повысить спрос на данное приложение среди туристов может, встроенный в визуализатор, механизм поворота 3D модели, что предоставит гостям крымского полуострова возможность рассмотреть, интересующую их, достопримечательность со всех сторон. Отличным решением также будет добавление краткой исторической сводки для каждой достопримечательности.

Финальным вариантом приложения будет программа, которая, помимо визуализации достопримечательностей и информации о них, будет иметь интеграцию с картами, для построения маршрута до выбранной достопримечательности, а также встроенного аудиогида с широкой информацией об историческом наследии Республики Крым. Одним из основных преимуществ данного приложения должна быть доступность, в том числе и для людей с ограниченными возможностями.

Список использованных источников

1. 3D редакторы, плюсы и минусы / Habr [сайт]. – URL: https://habr.com/ru/post/136350/ (дата обращения: 22.04.2020).
2. A guide to Google's Cloud Vision / Richard Mattka ; Creative Blog. – URL: https://www.creativebloq.com/how-to/a-guide-to-googles-cloud-vision/ (дата обращения: 18.04.2020).
3. Augmented reality with Python and OpenCV (part 1) / Bites of code [сайт]. – URL: https://bitesofcode.wordpress.com/2017/09/12/augmented-reality-with-python-and-opencv-part-1/ (дата обращения: 22.04.2020).
4. Bus Times London – TfL timetable and travel info : программный продукт / Google Play [сайт]. – URL: https://play.google.com/store/apps/details?id=com.mxdata.buslondon&hl=en/ (дата обращения: 13.04.2020).
5. CoMNIST / Kaggle [сайт]. – URL: https://www.kaggle.com/gregvial/comnist (дата обращения: 20.04.2020).
6. CuneiForm / Wikipedia [сайт]. – URL: https://ru.wikipedia.org/wiki/CuneiForm/ (дата обращения: 18.04.2020).
7. Google Переводчик / Google Play [сайт]. – URL: https://play.google.com/store/apps/details?id=com.google.android.apps.translate (дата обращения: 13.04.2020).
8. HoloAnatomy : программный продукт / Microsoft [сайт]. – URL: https://www.microsoft.com/ru-ru/p/holoanatomy/9nblggh4ntd3/ (дата обращения: 13.04.2020).
9. Niblack, W. An Introduction to Digitall Image Processing. Englewood Cliffs: PrenticeHall / W. Niblack. – 1986. – 215 p.
10. Ntirogiannis, K. An objective evaluation methodology for document image binarization techniques in DAS / K. Ntirogiannis, B. Gatos, I. Pratikakis. – 2008. – P. 217–224.
11. Optical character recognition / Wikipedia [сайт]. – URL: https://en.wikipedia.org/wiki/Optical_character_recognition/ (дата обращения: 13.04.2020).
12. Say hej to IKEA Place / Ikea [сайт] – URL: https://www.ikea.com/au/en/customer-service/mobile-apps/say-hej-to-ikea-place-pub1f8af050 (дата обращения: 13.04.2020).

13. Smith, R. An Overview of the Tesseract OCR Engine / Ray Smith ; Google Inc. – URL: https://storage.googleapis.com/pub-tools-public-publication-data/pdf/33418.pdf (дата обращения: 18.04.2020).
14. VR и AR в медицине / Яндекс Дзен [сайт]. – URL: https://zen.yandex.ru/media/id/5ad5fbd76104934709d6f592/vr-i-ar-v-medicine-5b17aa5d83090530bc94c7c4/ (дата обращения: 13.04.2020).
15. Vuforia / Wikipedia [сайт]. – URL: https://en.wikipedia.org/wiki/Vuforia_Augmented_Reality_SDK/ (дата обращения: 21.04.2020).
16. Глубокое обучение (Deep Learning): краткий туториал / Neurohive [сайт]. – URL: https://neurohive.io/ru/osnovy-data-science/glubokoe-obuchenie-deep-learning-kratkij-tutorial/ (дата обращения: 20.04.2020).
17. Дополненная реальность в маркетинге: 15+ примеров использования / Е. Самородских ; TexTerra – агентство комплексного интернет-маркетига. – URL: https://texterra.ru/blog/dopolnennaya-realnost-v-marketinge-primery-ispolzovaniya.html (дата обращения: 13.04.2020).
18. Исрафилов, Х. С. Исследование методов бинаризации изображений : статья / Х. С. Исрафилов – Москва : МГТУ им. Н.Э. Баумана, 2017. – 7 с.
19. Логический подход к искусственному интеллекту / под ред. Г. П. Гаврилова – Москва : Мир, 1998 – 493 с.
20. Методы распознавания текста / Habr [сайт]. – URL: https://habr.com/ru/post/220077/ (дата обращения: 14.04.2020).
21. Мир после Pokemon Go: как технология дополненной реальности изменит нашу жизнь / А. Сошников. – URL: https://www.bbc.com/russian/features-36898771 (дата обращения: 13.04.2020).
22. Николаев, Д. П. Критерии оценки качества в задаче автоматизированной настройки алгоритмов бинаризации : Труды ИСА РАН / Д. П. Николаев, А. А. Сараев. – Том 63. – 2013. – 10 с.
23. Обзор популярных AR-фреймворков / Lodoss Team – разработка мобильных приложений ; Habr [сайт]. – URL: https://habr.com/ru/company/lodoss/blog/358780/ (дата обращения: 20.04.2020).
24. Обзор самых популярных 3D редакторов / перевод Г. Абдуллина ; Videosmile [сайт]. – URL: https://videosmile.ru/lessons/read/421/ (дата обращения: 22.04.2020).

25. Оптическое распознавание символов / Studbooks [сайт]. – URL: https://studbooks.net/2016242/informatika/osnovnye_trudnosti_raspoznavaniya_simvolov/ (дата обращения: 13.04.2020).
26. Пасынков Д. А. Разработка приложения дополненной реальности : VII Международная интернет-конференция «Проблемы качества графической подготовки студентов в техническом вузе: традиции и инновации» / Пасынков Д. А. – Пермь, 2017. – С. 429–435.
27. Распознавание текста с помощью решений ABBYY – все гениальное просто для бизнеса / Комсомольская правда [сайт].– URL: https://www.kp.ru/guide/raspoznavanie-teksta.html (дата обращения: 13.04.2020).
28. Технология дополненной реальности AR / Увлекательная реальность – разработка мультимедийных продуктов. – URL: https://funreality.ru/technology/augmented_reality/ (дата обращения: 20.04.2020).
29. Тропченко, А. Ю. Методы вторичной обработки и распознавания изображений : учебное пособие / А. Ю. Тропченко, А. А. Тропченко. – Санкт-Петербург : Университет ИТМО, 2015. – С. 60–89.
30. Фор, А. Восприятие и распознавание образов. / А. Фор. – Москва : Машиностроение, 1989. – 272 с.
31. Фу, К. Структурные методы в распознавании образов / К. Фу. – Москва : Мир, 1977. – 320 с.
32. Эндрю, А. Искусственный интеллект / Под ред. Поспелова Д.А. – Москва : Мир, 1985. – 265 с.

Приложения

Приложение 1. Листинг скрипта *main.py*.

```
# import kivy packages
from kivy.app import App
from kivy.uix.widget import Widget
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.label import Label
from kivy.uix.image import Image
from kivy.clock import Clock
from kivy.graphics.texture import Texture
# import Recognition package
import pytesseract as tesseract
# import threading package
import continuous_threading as ct
# import other packages
from fuzzywuzzy import fuzz
# my packages
from src.objloader import *
from src.render import *

# Создание класса для хранения городов
class City():
    def __init__(self, image, model):
        self.img = cv2.imread(image,0)
        self.kp, self.desc = sift.detectAndCompute(self.img, None)
        self.model = OBJ(model, swapyz=True)

Simf = City('ref/Simf.jpg', 'models/Simf.obj')
Sevas = City('ref/Sevas.jpg', 'models/Sevas.obj')
Yalta = City('ref/Yalta.jpg', 'models/Yalta.obj')
Kerch = City('ref/Kerch.jpg', 'models/Kerch.obj')
Sudak = City('ref/Sudak.jpg', 'models/Sudak.obj')
th = ''

# Определение схожести строк
def similarity(s1, s2):
    if len(s1) < len(s2)/2:
        return 0
    low1 = s1.lower()
    low2 = s2.lower()
    match = fuzz.partial_ratio(low2, low1)
    return match/100


# Извлечение текста из изображения
def getTextWithTesseract(frame):
```

```
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    blur = cv2.GaussianBlur(gray,(5,5),0)
    edges = cv2.Canny(blur,100,200)

    # Для распознавания текста как одного слова
    custom_oem_psm_config = r'--oem 3 --psm 7 bazaar'
    # Поиск слова и разделение его на символы с координатами каждого
из них
    dirty      =      tesseract.image_to_string(edges,      lang="rus",
config=custom_oem_psm_config)
    # print(dirty)
    dirty = dirty.split('\n')
    allSymb = list(map((lambda x: list(x)), dirty))
    # Удаление не кириллических символов
    letters = list(map(lambda e: list(filter(lambda x: len(x) > 0 and
ord(x) in range(1040, 1103), e)), allSymb))

    text = ''.join(list(map(lambda x: ''.join(x), letters)))

    return text

class CamApp(App):

    def build(self):
        self.img1=Image()
        layout = BoxLayout(orientation="vertical")
        layout.add_widget(self.img1)
        # Захват кадра
        self.capture = cv2.VideoCapture(0)
        self.city = Simf
        _, self.goodImg = self.capture.read()
        Clock.schedule_interval(self.update, 1.0/60.0)
        th = ct.PeriodicThread(1.0, self.recognition)
        th.start()
        return layout

    def update(self, dt):
        _, frame = self.capture.read()

        frame = pasteModelIntoFrame(self.city, frame, self.goodImg);

        buf1 = cv2.flip(frame, 0)

        width = self.img1.size[0]
        height = int(frame.shape[0]*width/frame.shape[1])
        dim = (width, height)
        resized    =    cv2.resize(buf1,    dim,    interpolation    =
cv2.INTER_AREA)
```

```
        buf = resized.tostring()
        texture1       =       Texture.create(size=(resized.shape[1],
resized.shape[0]), colorfmt='bgr')
        texture1.blit_buffer(buf, colorfmt='bgr', bufferfmt='ubyte')

        self.img1.texture = texture1

    def recognition(self):
        _, self.goodImg = self.capture.read()

        text = getTextWithTesseract(self.goodImg)
        text = text.lower()
        print(text)
        if similarity(text, "симферополь") > 0.7:
            self.city = Simf
        elif similarity(text, "севастополь") > 0.7:
            self.city = Sevas
        elif similarity(text, "керчь") > 0.7:
            self.city = Kerch
        elif similarity(text, "судак") > 0.7:
            self.city = Sudak
        elif similarity(text, "ялта") > 0.7:
            self.city = Yalta

if __name__ == '__main__':
    CamApp().run()
```

Приложение 2. Листинг скрипта *render.py*.

```
import cv2
import numpy as np
import math

sift = cv2.xfeatures2d.SIFT_create()
camera_parameters = np.array([[800, 0, 320], [0, 800, 240], [0, 0,
1]])
index_params = dict(algorithm = 0, trees = 5)
search_params = dict()
flann = cv2.FlannBasedMatcher(index_params, search_params)
MIN_MATCH_COUNT = 15

def render(img, obj, projection, model):
    """
    Рендер 3D модели на кадр
    """
    vertices = obj.vertices
    scale_matrix = np.eye(3)
```

```
    h, w = model.shape

    for face in obj.faces:
        face_vertices = face[0]
        points = np.array([vertices[vertex - 1] for vertex in
face_vertices])
        points = np.dot(points, scale_matrix)
        # Визуализация модели в середине опорной поверхности.
        # Для этого точки модели должны быть смещены
        points = np.array([[p[0] + w / 2, p[1] + h / 2, p[2]] for p
in points])
        distance = np.array([p[1] for p in points])
        dst = cv2.perspectiveTransform(points.reshape(-1, 1, 3),
projection)
        imgpts = np.int32(dst)

        positive = list(map(lambda x: abs(x), distance))
        color = 128*np.mean(distance)/max(positive)
        color = tuple([round(color)] * 3)
        cv2.fillConvexPoly(img, imgpts, color)

    return img

def projection_matrix(camera_parameters, homography):
    """
    Вычисление матрицы 3D-проекции из калибровочной матрицы камеры и
расчетной гомографии.
    """
    # Вычисление поворота и смещения по осям x и y
    homography = homography * (-1)
    rot_and_transl = np.dot(np.linalg.inv(camera_parameters),
homography)
    col_1 = rot_and_transl[:, 0]
    col_2 = rot_and_transl[:, 1]
    col_3 = rot_and_transl[:, 2]
    # Нормирование векторов
    l = math.sqrt(np.linalg.norm(col_1, 2) * np.linalg.norm(col_2,
2))
    rot_1 = col_1 / l
    rot_2 = col_2 / l
    translation = col_3 / l
    # Вычисление ортогонального базиса
    c = rot_1 + rot_2
    p = np.cross(rot_1, rot_2)
    d = np.cross(c, p)
    rot_1 = np.dot(c / np.linalg.norm(c, 2) + d / np.linalg.norm(d,
2), 1 / math.sqrt(2))
```

```
    rot_2 = np.dot(c / np.linalg.norm(c, 2) - d / np.linalg.norm(d,
2), 1 / math.sqrt(2))
    rot_3 = np.cross(rot_1, rot_2)
    # Вычисление 3D-проекции модели на текущий кадр
    projection = np.stack((rot_1, rot_2, rot_3, translation)).T
    return np.dot(camera_parameters, projection)

def sort_points(faces, vertices):
    all_points = []
    for face in faces:
        face_vertices = face[0]
        points = np.array([vertices[vertex - 1] for vertex in
face_vertices])
        all_points.append(points)

    sorted_points = sorted(all_points, key=lambda p:
(p[0][1]+p[1][1]+p[2][1])/3)

    return sorted_points

# Проекция 3D модели в кадр
def pasteModelIntoFrame(city, frame, goodImg):
    kp1 = city.kp
    desc1 = city.desc
    final_image = frame
    grayframe = cv2.cvtColor(goodImg, cv2.COLOR_BGR2GRAY)
    kp2, desc2 = sift.detectAndCompute(grayframe, None)
    if desc2 is not None:
        # Находим по 2 ближайших дескриптора для каждой точки
        # Два раза: маркер к картинке и обратно (они будут разными)
        matches1to2 = flann.knnMatch(desc1, desc2, k=2)
        matches2to1 = flann.knnMatch(desc2, desc1, k=2)
        # Выкидываем точки с менее чем двумя соответствиями
        matches1to2 = [x for x in matches1to2 if len(x) == 2]
        matches2to1 = [x for x in matches2to1 if len(x) == 2]
        # Выкидываем точки, в которых не сильно уверены
        ratio = 0.6
        good1to2 = [m for m,n in matches1to2 if m.distance < ratio *
n.distance]
        good2to1 = list([m for m,n in matches2to1 if m.distance <
ratio * n.distance])
        # Выкидываем несимметричные соответствия
        good = []
        for m in good1to2:
            for n in good2to1:
                if m.queryIdx == n.trainIdx and n.queryIdx ==
m.trainIdx:
                    good.append(m)
```

```
        if len(good) > MIN_MATCH_COUNT:
            query_pts = np.float32([kp1[m.queryIdx].pt for m in
good]).reshape(-1, 1, 2)
            train_pts = np.float32([kp2[m.trainIdx].pt for m in
good]).reshape(-1, 1, 2)
            matrix, mask = cv2.findHomography(query_pts, train_pts,
cv2.RANSAC, 5.0)
            matches_mask = mask.ravel().tolist()
            # if matrix is not None:
            #     # Perspective transform
            #     h, w = city.img.shape
            #       pts = np.float32([[0, 0], [0, h], [w, h], [w,
0]]).reshape(-1, 1, 2)
            #     dst = cv2.perspectiveTransform(pts, matrix)
            #             final_image = cv2.polylines(final_image,
[np.int32(dst)], True, (255, 0, 0), 3)
            if matrix is not None:
                print(matrix)
                # Получение матрицы 3D-проекции из параметров матрицы
гомографии и камеры
                projection = projection_matrix(camera_parameters,
matrix)
                # Проектирование модели
                final_image = render(frame, city.model, projection,
city.img)

    return final_image
```

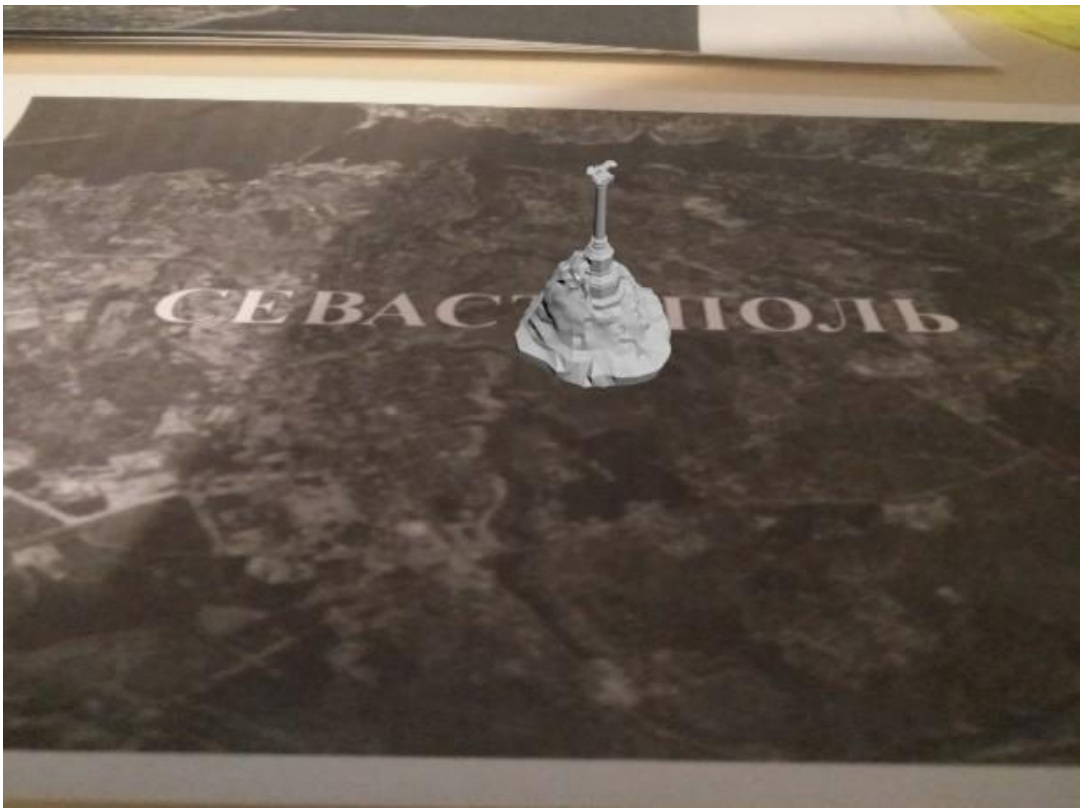
Приложение 3. Листинг скрипта *objloader.py*.

```
class OBJ:
    def __init__(self, filename, swapyz=False):
        self.vertices = []
        self.normals = []
        self.texcoords = []
        self.faces = []
        material = None
        for line in open(filename, "r"):
            if line.startswith('#'): continue
            values = line.split()
            if not values: continue
            if values[0] == 'v':
                v = list(map(float, values[1:4]))
                if swapyz:
                    v = v[0], v[2], v[1]
                self.vertices.append(v)
            elif values[0] == 'vn':
```

```
                v = list(map(float, values[1:4]))
                if swapyz:
                    v = v[0], v[2], v[1]
                self.normals.append(v)
            elif values[0] == 'vt':
                self.texcoords.append(list(map(float, values[1:3])))
            elif values[0] in ('usemtl', 'usemat'):
                material = values[1]
            elif values[0] == 'mtllib':
                self.mtl = MTL(values[1])
            elif values[0] == 'f':
                face = []
                texcoords = []
                norms = []
                for v in values[1:]:
                    w = v.split('/')
                    face.append(int(w[0]))
                    if len(w) >= 2 and len(w[1]) > 0:
                        texcoords.append(int(w[1]))
                    else:
                        texcoords.append(0)
                    if len(w) >= 3 and len(w[2]) > 0:
                        norms.append(int(w[2]))
                    else:
                        norms.append(0)
                #    self.faces.append((face,    norms,    texcoords,
material))
                self.faces.append((face, norms, texcoords))
    def __str__(self):
        print(self.vertices)
        print(self.normals)
        print(self.texcoords)
        print(self.faces)
        return "end"
```

Приложение 4. Результат, Симферополь.

Приложение 5. Результат, Севастополь.



Приложение 6. Результат, Керчь.

Приложение 7. Результат, Судак.

Приложение 8. Результат, Ялта.