



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ

«Информатика и системы управления»

КАФЕДРА

«Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К КУРСОВОЙ РАБОТЕ
НА ТЕМУ:

***«Определение наиболее вероятного соответствия объекта,
изображённого на двумерном снимке, с одним из объектов из
списка стандартных тел»***

Студент ИУ7-55Б
(Группа)

Руководитель курсовой работы

(Подпись, дата) Бугаенко А. П.
(Фамилия И. О.)

(Подпись, дата) Новик Н. В.
(Фамилия И. О.)

2022 г.

Содержание

Введение	7
1 Аналитический раздел	9
1.1 Анализ предметной области	9
1.2 Формализация объектов сцены	10
1.3 Анализ виртуальной камеры	10
1.4 Вывод из анализа виртуальной камеры	11
1.5 Анализ способа представления поверхности	11
1.5.1 Способы представления поверхностной модели	12
1.6 Вывод из анализа способов представления поверхности модели	13
1.7 Анализ алгоритмов удаления невидимых линий и поверхностей	13
1.8 Вывод из анализа алгоритмов удаления невидимых линий и поверхностей	15
1.9 Анализ методов закрашивания	15
1.10 Вывод из анализа методов закрашивания	17
1.11 Анализ алгоритмов машинного обучения для определения наиболее вероятного соответствия объекта, изображённого на двумерном снимке, с одним из объектов из списка стандартных тел	17
1.12 Вывод из анализа алгоритмов машинного обучения для восстановления трёхмерной модели из двумерного изображения	18
1.13 Описание трёхмерных преобразований	19
1.13.1 Описание операции перемещения	19
1.13.2 Описание операции масштабирования	19
1.13.3 Описание операции поворота	20
1.14 Описание преобразования пространства сцены в пространство виртуальной камеры	22

1.15	Описание требований к разрабатываемому программному обеспечению	22
1.15.1	Описание входных и выходных данных	22
1.15.2	Описание функциональных требований к программному обеспечению	23
1.16	Вывод из аналитического раздела	24
2	Конструкторский раздел	25
2.1	Описание структуры программного обеспечения	25
2.1.1	Общая структура программного обеспечения	25
2.1.2	Диаграмма классов	25
2.2	Описание используемых алгоритмов	25
2.2.1	Общий алгоритм отрисовки	26
2.2.2	Алгоритм отсекаания невидимой части треугольного полигона	27
2.2.3	Алгоритм z-буфера	29
2.3	Описание архитектуры используемой свёрточной нейросети . .	30
2.3.1	Алгоритм обучения свёрточной нейросети	30
2.3.2	Алгоритм определения трёхмерного объекта, наиболее вероятно являющимся объектом изображённым на снимке	31
2.4	Описание используемых структур данных	31
2.4.1	Трёхмерный вектор	31
2.4.2	Треугольный полигон	31
2.4.3	Матрица	32
2.4.4	Трёхмерный объект	32
2.4.5	Камера	32
2.4.6	Освещение	32
2.4.7	Сцена	32
2.5	Вывод из конструкторского раздела	33
3	Технологический раздел	34
3.1	Выбор языка программирования и среды разработки	34
3.2	Листинги реализаций основных алгоритмов	35
3.2.1	Реализация общего алгоритма отрисовки	35

3.2.2	Реализация алгоритма отсекаания невидимой части треугольного полигона	38
3.2.3	Реализация алгоритма z-буфера	41
3.2.4	Реализация алгоритма обучения свёрточной нейросети .	45
3.2.5	Реализация алгоритма определения трёхмерного объекта, наиболее вероятно являющимся объектом изображённым на снимке	48
3.3	Сведения о модулях программы	49
3.3.1	Описание данных, использующихся при обучении свёрточной нейросети	50
3.4	Информация, необходимая для сборки и запуска разработанного программного обеспечения	50
3.4.1	Сборка программного обеспечения	50
3.4.2	Запуск программного обеспечения	51
3.5	Интерфейс разработанного программного обеспечения	51
3.5.1	Общие элементы интерфейса программы	51
3.5.2	Интерфейс параметров модели	52
3.5.3	Интерфейс параметров камеры	53
3.5.4	Интерфейс параметров освещения	54
3.5.5	Интерфейс синтеза изображения	54
3.5.6	Интерфейс определения наиболее вероятного соответствия объекта, изображённого на двумерном снимке	55
3.6	Вывод из технологического раздела	55
4	Экспериментальный раздел	56
4.1	Технические характеристики	56
4.2	Исследование зависимости скорости работы программы от количества полигонов в загружаемой модели	56
4.2.1	Цель эксперимента по исследованию зависимости скорости работы алгоритма отрисовки от количества полигонов в просматриваемой модели	57
4.2.2	Описание эксперимента по исследованию зависимости скорости работы алгоритма отрисовки от количества полигонов в просматриваемой модели	57

4.2.3	Результат эксперимента по исследованию зависимости скорости работы алгоритма отрисовки от количества полигонов в просматриваемой модели	57
4.3	Исследование эффективности работы разработанного алгоритма определения трёхмерного объекта, наиболее вероятно являющимся объектом изображённым на снимке . . .	58
4.3.1	Цель эксперимента по исследованию эффективности работы разработанного алгоритма определения трёхмерного объекта, наиболее вероятно являющимся объектом изображённым на снимке	59
4.3.2	Описание эксперимента по исследованию эффективности работы разработанного алгоритма определения трёхмерного объекта, наиболее вероятно являющимся объектом изображённым на снимке	59
4.3.3	Результат эксперимента по исследованию эффективности работы разработанного алгоритма определения трёхмерного объекта, наиболее вероятно являющимся объектом изображённым на снимке	59
4.4	Вывод из экспериментального раздела	60
Заключение		62
Список использованных источников		64
Приложение А		65
Приложение Б		66

Введение

В современном мире компьютерная графика используется во многих сферах человеческой деятельности для визуализации различного вида данных, включающих в себя моделирование, компьютерные игры и системы автоматизированного проектирования [1]. Одной из основных областей применения компьютерной графики является построение изображений трёхмерных объектов. Данная ветвь машинной графики, или 3D-графика, используется сегодня для решения широкого набора задач. Очевидно, что в любой задаче построения изображения трёхмерной сцены используются трёхмерные объекты, создание которых является затратным по времени, поскольку в большей части случаев модель создаётся человеком в ручную при помощи специального программного обеспечения.

Одним из способов снизить время, затрачиваемое на создание модели, является автоматизация создания трёхмерной модели, что может быть реализовано при помощи технологий машинного обучения. Одной из возможных реализаций такого способа является создание алгоритма, позволяющего подобрать модель в соответствии с изображением этой модели. Однако для обучения модели нужны данные, в связи с чем возникает необходимость создания редактора трёхмерных объектов, позволяющего генерировать данные для обучения модели.

Целью данной работы является создание программного обеспечения, которое позволило бы загружать, просматривать, редактировать масштаб, позицию и вращение трёхмерной модели, синтезировать изображение трёхмерной модели, и определять наиболее вероятное соответствие объекта, изображённого на двумерном снимке, с одним из объектов из списка стандартных тел.

Для достижения поставленной цели необходимо решить следующие задачи:

- исследовать подходы к синтезу изображений;
- сравнить существующие решения;
- описать используемые при разработке ПО алгоритмы;
- определить средства программной реализации;
- реализовать алгоритмы отрисовки сцены;
- реализовать алгоритмы генерации изображения;
- реализовать алгоритм позволяющий определять наиболее вероятное соответствие объекта, изображённого на двумерном снимке, с одним из объектов из списка стандартных тел;
- исследовать производительность алгоритма отрисовки для различных объектов;
- исследовать эффективность работы алгоритма восстановления трёхмерного объекта из его двумерного снимка.

1 Аналитический раздел

В данном разделе проводится анализ предметной области, анализ подходов к синтезу изображений трёхмерного объекта, осуществляется описание сцены, трёхмерных объектов на сцене, проводится сравнительный анализ различных алгоритмов и обзор существующих решений.

1.1 Анализ предметной области

В данной работе рассматривается предметная область трёхмерной компьютерной графики. Элементами данной области являются заказчики, формирующие задание на создание определённого трёхмерного объекта и передающие его команде специалистов по трёхмерной компьютерной графике для выполнения, команда специалистов по компьютерной графике, обладающих необходимыми умениями для того, чтобы создавать трёхмерные объекты, удовлетворяющие требованиям заказчика, специальные программы, позволяющие создавать и изменять различные трёхмерные объекты и трёхмерные объекты, являющиеся продуктом работы команды специалистов.

Специфика цикла разработки трёхмерных объектов в данной предметной области состоит в том, что создание трёхмерных объектов является времяёмким процессом из-за необходимости создания трёхмерного объекта практически вручную на основе либо текстового описания, либо двумерных изображений того, каким образом должен выглядеть создаваемый трёхмерный объект.

На основе проведённого выше анализа предметной области можно предложить следующий проект трёхмерного редактора, автоматизирующий работу команды специалистов по трёхмерной компьютерной графике в рассматриваемой области. Трёхмерная модель будет выбираться из некоего списка заранее заданных моделей на основе двумерного изображения, на котором находится двумерное изображение трёхмерного объекта, который

нужно создать. Использование данного приведёт к значительному снижению временных затрат на разработку трёхмерного объекта, поскольку работа команды специалистов по компьютерной графике сведётся к доведению модели до состояния, соответствующему техническому заданию от заказчика, а не созданию модели с нуля.

Таким образом основными задачами, выполняемыми данных проектом, являются создание трёхмерного редактора, позволяющего просматривать трёхмерные объекты и алгоритма, позволяющего определить наиболее вероятное соответствия объекта, изображённого на двумерном снимке, с одним из объектов из списка стандартных тел.

1.2 Формализация объектов сцены

В компьютерной графике сцена является объектом, описывающим трёхмерное пространство и все объекты, находящиеся в данном трёхмерном пространстве.

В реализуемом программном обеспечении сцена содержит компоненты следующих типов:

- источник света — задаёт овещение сцены;
- трёхмерный объект — трёхмерная модель, находящаяся на сцене;
- камера — виртуальная камера, позволяющая просматривать трёхмерные объекты, находящиеся на сцене и делать их снимки.

1.3 Анализ виртуальной камеры

Виртуальная камера - объект сцены трёхмерной виртуальной среды, эмулирующий камеру в трёхмерном пространстве сцены и позволяющий получать двумерный снимок (проекцию) сцены.

3D-проекция (или графическая проекция) — это метод проектирования, используемый для отображения трехмерного (3D) объекта на двумерной (2D) поверхности. Эти проекции основаны на визуальной перспективе и анализе аспектов, чтобы спроецировать сложный объект для возможности просмотра на более простой плоскости [1].

Существует два основных вида проецирования:

- параллельная проекция;
- перспективная проекция.

Параллельная проекция — это проекция объекта в трёхмерном пространстве на неподвижную плоскость, известную как плоскость проекции или плоскость изображения, где лучи, известные как линии обзора или проекционные линии, параллельны друг другу.

Перспективная проекция — это приблизительное представление, обычно на плоской поверхности, изображения таким, каким оно видится глазу.

В случае виртуальной камеры для трёхмерной виртуальной среды будет использоваться перспективная проекция, поскольку она содержит больше информации о том, каким образом трёхмерный объект расположен в пространстве.

1.4 Вывод из анализа виртуальной камеры

На основе результатов, полученных при анализе виртуальной камеры, сделан вывод о том, что для решения поставленных задач наилучшим решением будет использование при реализации виртуальной камеры перспективная проекции, поскольку она содержит больше информации о том, каким образом трёхмерный объект расположен в пространстве.

1.5 Анализ способа представления поверхности

Трёхмерная модель в пространстве сцены представляется тремя основными способами:

- каркасный — модель представляется в виде множества вершин, связанных между собой рёбрами;
- поверхностный — модель представляется в виде множества поверхностей;
- объёмный — модель формируется из элементарных объектов с использованием логических операций объединения, вычитания, пересечения.

В нашем случае наиболее подходящий способ описания модели — поверхностный, так как каркасные модели не обладают достаточным

количеством деталей необходимых для восстановления модели по её снимку. Объёмная модель не используется, так как алгоритм работает не с объёмом модели, а с её поверхностью.

1.5.1 Способы представления поверхностной модели

При использовании поверхностной модели необходимо определить, каким образом задаётся поверхность модели. Существует два основных способа представления поверхности модели:

- Параметрическое представление — поверхность определяется некой функцией, зависящей от одного или нескольких параметров.
- Полигональная сетка — в этом случае поверхность задаётся совокупностью вершин, рёбер и граней, которые определяют внешний вид объекта.

Наиболее подходящим способом представления поверхности модели является полигональная сетка в силу его универсальности и повсеместного применения в современном 3D моделировании. Параметрическое представление не используется, поскольку в основном данный метод используется для задания в трёхмерном пространстве тел вращения. Этот метод не подходит для представления поверхности, так как представление в сцене объектов, не являющихся объектами вращения, излишне усложнено.

При использовании полигональной сетки необходимо определить, каком образом задаётся её структура. Существует четыре основных способа представления полигональной сетки:

- Вершинное представление — полигональная сетка представляется как список, в котором для каждой вершине ставятся в соответствие вершины, с которыми она соединена рёбрами.
- Таблица углов — в данном случае вершины хранятся в предопределённой таблице.
- Список граней — объект представляется как множество граней и вершин.
- Список полигонов — объект представляется как множество элементарных поверхностей, формирующих поверхность модели.

Наиболее подходящим способом представления полигональной сетки является список треугольных полигонов, в совокупности составляющий поверхность модели, поскольку это позволит исключить из алгоритмов

операции получения нужной вершины при работах с более сложными структурами данных. Вершинное представление не используется, поскольку оно затратно по времени, так как для того, чтобы сформировать грани приходится проходить под данным несколько раз. Таблица углов не используется, поскольку при изменении данных затратно по времени.

1.6 Вывод из анализа способов представления поверхности модели

На основе полученного в результате анализа способов представления поверхности моделей сделан вывод о том, что для реализации представления поверхности наиболее подходящим является поверхностный способ описания модели, представленный как полигональная сетка, задающаяся списком треугольных полигонов.

1.7 Анализ алгоритмов удаления невидимых линий и поверхностей

При выборе алгоритма удаления невидимых линий и поверхностей нужно учитывать особенности поставленной задачи. Одной из таких особенностей является то, что реалистическое изображение не обязательно нужно для работы алгоритма восстановления трёхмерного объекта из двумерного снимка, поскольку распознавание основывается на общей геометрии объекта, а не на деталях поверхности объекта. При этом не использование сложных алгоритмов не предпочтительно, поскольку это приведёт к излишним затратам по времени и как следствие — к замедлению работы программы.

Алгоритм трассировки лучей — при построении изображения луч посылается в заданном направлении для оценки приходящей оттуда световой энергии. Эта энергия определяется освещённостью первой поверхности, встретившейся на пути луча. Из каждого источника света выпускаются лучи во все стороны. При пересечении луча с поверхностью получают отражённый и преломлённый лучи, а если моделировать диффузное освещение, то и пучок лучей, равномерно распространяющийся во все стороны. Данный метод имеет

наивысшую степень реализма и вместе с этим высокую сложность реализации и затрат ресурсов при работе программы [4].

Алгоритм Робертса — Алгоритм Робертса представляет собой первое известное решение задачи об удалении невидимых линий. Это математически элегантный метод, работающий в объектном пространстве. Алгоритм прежде всего удаляет из каждого тела те ребра или грани, которые экранируются самим телом. Пусть F — некоторая грань многогранника. Плоскость, несущая эту грань, разделяет пространство на два подпространства. Назовем положительным то из них, в которое смотрит внешняя нормаль к грани. Если точка наблюдения в положительном подпространстве, то грань лицевая, в противном случае грань является нелицевой. Если многогранник выпуклый, то удаление всех нелицевых граней полностью решает задачу визуализации с удалением невидимых граней. Невыпуклые тела должны быть разбиты на выпуклые части. В этом алгоритме выпуклое многогранное тело с плоскими гранями должно представиться набором пересекающихся плоскостей [4].

Алгоритм Варнака — алгоритм Варнака является ещё одним примером алгоритма, основанного на разбиении картинной плоскости на части, для каждой из которых исходная задача может быть решена достаточно просто. Разобьём видимую часть картинной плоскости на 4 равные части. В случаях, когда часть полностью накрывается проекцией ближайшей грани и часть не накрывается проекцией ни одной грани, вопрос о закрашивании соответствующей части решается тривиально. В случае, когда ни одно из этих условий не выполнено, данная часть разбивается на 4 части, для каждой из которых проверяется выполнение этих условий, и так далее. Очевидно, что разбиение имеет смысл проводить до тех пор, пока размер части больше, чем размер пиксела [5].

Алгоритм, использующий Z-буфер — данный алгоритм является одним из простейших алгоритмов удаления невидимых поверхностей. Работает этот алгоритм в пространстве изображения. Идея Z-буфера является простым обобщением идеи о буфере кадра. Буфер кадра используется для запоминания атрибутов (интенсивности) каждого пиксела в пространстве изображения, Z-буфер — это отдельный буфер глубины, используемый для запоминания координаты z или глубины каждого видимого пиксела в пространстве изображения. В процессе работы глубина или значение z

каждого нового пиксела, который нужно занести в буфер кадра, сравнивается с глубиной того пиксела, который уже занесен в Z-буфер. Если это сравнение показывает, что новый пиксел расположен впереди пиксела, находящегося в буфере кадра, то новый пиксел заносится в этот буфер и, кроме того, производится корректировка Z-буфера новым значением z . Если же сравнение дает противоположный результат, то никаких действий не производится. По сути, алгоритм является поиском по x и y наибольшего значения функции $z(x, y)$ [4].

Поскольку в данной задаче важна скорость построения изображения для динамической отрисовки трёхмерной сцены и не требуется построение реалистического изображения, используется алгоритм z-буфера.

1.8 Вывод из анализа алгоритмов удаления невидимых линий и поверхностей

На основе проведённого анализа алгоритмов удаления невидимых линий и поверхностей сделан вывод о том, что для реализации алгоритма удаления невидимых линий и поверхностей наиболее подходит алгоритм z-буфера, поскольку в данной задаче важна скорость построения изображения для динамической отрисовки трёхмерной сцены и не требуется построение реалистического изображения.

1.9 Анализ методов закрашивания

В компьютерной графике для расчета освещенности граней объектов применяется цветовая модель RGB. Интенсивность отраженного света точек объектов вычисляют отдельно для каждой из трех составляющих цветовых компонент, а затем объединяют в результирующую тройку цветов. При расчете освещенности граней применяют следующие типы освещения и отражения света от поверхностей:

- рассеянное;
- диффузное;
- зеркальное.

Матовые поверхности обладают свойством диффузного отражения, т. е. равномерного по всем направлениям рассеивания света. Поэтому кажется, что поверхности имеют одинаковую яркость независимо от угла обзора. Для таких поверхностей справедлив закон косинусов Ламберта, устанавливающий соответствие между количеством отраженного света и косинусом угла θ между направлением на точечный источник света интенсивности I_p и нормалью к поверхности. При этом количество отраженного света не зависит от положения наблюдателя. Освещенность рассеянным светом вычисляется по формуле 1.1.

$$I_d = I_p \cdot k_d \cdot \cos\theta. \quad (1.1)$$

Значение коэффициента диффузного отражения k_d является константой в диапазоне $(0, 1)$ и зависит от материала [3].

Закраска по Гуро — алгоритм закраски поверхности по Гуро позволяет устранить дискретность изменения интенсивности. Процесс закраски по методу Гуро осуществляется в четыре этапа:

1. вычисляются нормали ко всем полигонам;
2. определяются нормали в вершинах путем усреднения нормалей по всем полигональным граням, которым принадлежит вершина;
3. используя нормали в вершинах и применяя произвольный метод закраски, вычисляются значения интенсивности в вершинах;
4. каждый многоугольник закрашивается путем линейной интерполяции значений интенсивностей в вершинах сначала вдоль каждого ребра, а затем и между ребрами вдоль каждой сканирующей строки.

Закраска по Фонгу — В методе закраски Фонга используется интерполяция вектора нормали к поверхности вдоль видимого интервала на сканирующей строке внутри многоугольника, а не интерполяция интенсивности. Интерполяция выполняется между начальной и конечной нормалью, которые сами тоже являются результатами интерполяции вдоль ребер многоугольника между нормалью в вершинах. Нормали в вершинах, в свою очередь, вычисляются так же, как в методе закраски, построенном на основе интерполяции интенсивности. В каждом пикселе вдоль сканирующей строки новое значение интенсивности вычисляется с помощью любой модели закраски. Заметные улучшения по сравнению с интерполяцией интенсивности наблюдаются в случае использования модели с учетом зеркального отражения,

т. к. при этом более точно воспроизводятся световые блики. Однако даже если зеркальное отражение не используется, интерполяция векторов нормали приводит к более качественным результатам, чем интерполяция интенсивности, поскольку аппроксимация нормали в этом случае осуществляется в каждой точке [3].

В данной задаче для реализации закрашивания наиболее подходит метод простой диффузной закраски с использованием закона Ламберта, поскольку редактор должен работать в режиме реального времени и не требуется расчёт отражений и бликов при построении изображения модели.

1.10 Вывод из анализа методов закрашивания

На основе проведённого анализа методов закрашивания сделан вывод о том, что в качестве реализации метода закрашивания наиболее подходит метод простой диффузной закраски с использованием закона Ламберта, поскольку редактор должен работать в режиме реального времени и не требуется расчёт отражений и бликов при построении изображения модели.

1.11 Анализ алгоритмов машинного обучения для определения наиболее вероятного соответствия объекта, изображённого на двумерном снимке, с одним из объектов из списка стандартных тел

Машинное обучение — процесс, в котором точность работы компьютера повышается по мере сбора данных и извлечения из них информации [7].

Нейронная сеть — математическая модель, а также её программное или аппаратное воплощение, построенная по принципу организации и функционирования биологических нейронных сетей — сетей нервных клеток живого организма [8].

Свёрточная нейросеть — специальная архитектура искусственных нейронных сетей нацеленная на эффективное распознавание образов.

Данная архитектура использует некоторые особенности зрительной коры, в которой существуют простые клетки, реагирующие на прямые линии под разными углами, и сложные клетки, реакция которых связана с активацией определённого набора простых клеток. Таким образом, идея свёрточных нейронных сетей заключается в чередовании свёрточных слоёв и слоёв подвыборки. Структура сети — однонаправленная, многослойная. Для обучения используются стандартные методы, чаще всего метод обратного распространения ошибки. Основное отличие от многослойного перцептрона является наличие операции свёртки, суть которой заключается в том, что каждый фрагмент изображения умножается на матрицу свёртки поэлементно, а результат суммируется и записывается в аналогичную позицию выходного изображения. Работа свёрточной нейронной сети обычно интерпретируется как переход от конкретных особенностей изображения к более абстрактным деталям, и далее к ещё более абстрактным деталям вплоть до выделения понятий высокого уровня. При этом сеть самонастраивается и вырабатывает сама необходимую иерархию абстрактных признаков, фильтруя маловажные детали и выделяя существенное [9].

В данной работе используется свёрточная нейронная сеть, поскольку одной из задач в данной работе является создание программного обеспечения, способного восстановить трёхмерную модель из двумерного изображения. С помощью свёрточных нейронных сетей, позволяющих извлекать абстрактные признаки из изображений, становится возможно получить из изображения информацию, позволяющую решить данную задачу.

1.12 Вывод из анализа алгоритмов машинного обучения для восстановления трёхмерной модели из двумерного изображения

На основе проведённого анализа алгоритмов машинного обучения для восстановления трёхмерной модели из двумерного изображения сделан вывод о том, что в качестве реализации наиболее подходит свёрточная нейронная сеть.

1.13 Описание трёхмерных преобразований

Все трёхмерные преобразования являются результатом применения серии элементарных операций преобразования к какому-либо трёхмерному объекту. Данные элементарные преобразования включают в себя перемещение, масштабирование и поворот.

1.13.1 Описание операции перемещения

Рассмотрим операцию перемещения. Пусть координаты некоторой точки в трёхмерном пространстве задаются вектором $\vec{p} = \{p_x, p_y, p_z\}$, а вектор перемещения задаётся вектором $\vec{d} = \{d_x, d_y, d_z\}$, тогда координаты перемещённой точки в пространстве задаются вектором $\vec{p'}$, который вычисляется по формуле 1.2.

$$\vec{p'} = \begin{cases} p'_x = p_x + d_x \\ p'_y = p_y + d_y \\ p'_z = p_z + d_z \end{cases} \quad (1.2)$$

Формула 1.2 также может быть представлена виде матричного уравнения, где вектор координат умножается на матрицу коэффициентов. При использовании такого способа в вектор необходимо добавить четвёртую компоненту, равную 0, для того, чтобы размерности вектора и матрицы соответствовали для проведения операции умножения. В этом случае вектор $\vec{p'}$ будет вычисляться по формуле 1.3.

$$\vec{p'} = \begin{bmatrix} p_x & p_y & p_z & 0 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ d_x & d_y & d_z & 1 \end{bmatrix} \quad (1.3)$$

1.13.2 Описание операции масштабирования

Рассмотрим операцию масштабирования. Данная операция происходит относительно точки, называемой центром масштабирования. Пусть в этом

случае центр масштабирования - начало координат, координаты некоторой точки в трёхмерном пространстве задаются вектором $\vec{p} = \{p_x, p_y, p_z\}$, а вектор коэффициентов масштабирования точки задаётся вектором $\vec{s} = \{s_x, s_y, s_z\}$, тогда координаты масштабированной относительно центра координат точки в пространстве задаются вектором $\vec{p'}$, который вычисляется по формуле 1.4.

$$\vec{p'} = \begin{cases} p'_x = p_x \cdot s_x \\ p'_y = p_y \cdot s_y \\ p'_z = p_z \cdot s_z \end{cases} \quad (1.4)$$

Формула 1.4 также может быть представлена виде матричного уравнения, где вектор координат умножается на матрицу коэффициентов. При использовании такого способа в вектор необходимо добавить четвёртую компоненту, равную 0, для того, чтобы размерности вектора и матрицы соответствовали для проведения операции умножения. В этом случае вектор $\vec{p'}$ будет вычисляться по формуле 1.5.

$$\vec{p'} = \begin{bmatrix} p_x & p_y & p_z & 0 \end{bmatrix} \times \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1.5)$$

1.13.3 Описание операции поворота

Рассмотрим операцию поворота. Данная операция происходит относительно оси, называемой осью вращения. Рассмотрим вращение относительно осей OX, OY, OZ . Пусть координаты некоторой точки в трёхмерном пространстве задаются вектором $\vec{p} = \{p_x, p_y, p_z\}$, а угол поворота относительно оси равен ϕ . Тогда координаты повернутой относительно осей OX, OY, OZ точки в пространстве задаются векторами $\vec{p'_{OX}}, \vec{p'_{OY}}, \vec{p'_{OZ}}$, которые соответственно вычисляются по формулам 1.6, 1.7, 1.8.

$$\vec{p'_{OX}} = \begin{cases} p'_x = p_x \\ p'_y = p_y \cdot \cos\phi + p_z \cdot \sin\phi \\ p'_z = -p_y \cdot \sin\phi + p_z \cdot \cos\phi \end{cases} \quad (1.6)$$

$$\overrightarrow{p'_{OY}} = \begin{cases} p'_x = p_x \cdot \cos\phi - p_z \cdot \sin\phi \\ p'_y = p_y \\ p'_z = p_x \cdot \sin\phi + p_z \cdot \cos\phi \end{cases} \quad (1.7)$$

$$\overrightarrow{p'_{OZ}} = \begin{cases} p'_x = p_x \cdot \cos\phi + p_y \cdot \sin\phi \\ p'_y = -p_x \cdot \sin\phi + p_z \cdot \cos\phi \\ p'_z = p_z \end{cases} \quad (1.8)$$

Формулы 1.6-1.8 также могут быть представлены в виде матричного уравнения, где вектор координат умножается на матрицу коэффициентов. При использовании такого способа в вектор необходимо добавить четвёртую компоненту, равную 0, для того, чтобы размерности вектора и матрицы соответствовали для проведения операции умножения. В этом случае вектора $\overrightarrow{p'_{OX}}, \overrightarrow{p'_{OY}}, \overrightarrow{p'_{OZ}}$ соответственно вычисляются по формулам 1.9, 1.10, 1.11.

$$\overrightarrow{p'_{OX}} = \begin{bmatrix} p_x & p_y & p_z & 0 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi & 0 \\ 0 & \sin\phi & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1.9)$$

$$\overrightarrow{p'_{OY}} = \begin{bmatrix} p_x & p_y & p_z & 0 \end{bmatrix} \times \begin{bmatrix} \cos\phi & 0 & \sin\phi & 0 \\ 0 & 0 & 0 & 0 \\ -\sin\phi & 0 & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1.10)$$

$$\overrightarrow{p'_{OZ}} = \begin{bmatrix} p_x & p_y & p_z & 0 \end{bmatrix} \times \begin{bmatrix} \cos\phi & -\sin\phi & 0 & 0 \\ \sin\phi & \cos\phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1.11)$$

1.14 Описание преобразования пространства сцены в пространство виртуальной камеры

Для того, чтобы преобразовать сцену в пространство камеры, и при этом сохранить перспективу, необходимо координату каждой точки сцены умножить на матрицу перспективной проекции. Матрица проекции отображает заданный диапазон усеченной пирамиды в пространство отсечения, и при этом манипулирует w -компонентой каждой вершины таким образом, что чем дальше от наблюдателя находится вершина, тем больше становится это w -значение. После преобразования координат в пространство отсечения, все они попадают в диапазон от $-w$ до w (вершины, находящиеся вне этого диапазона, отсекаются).

Пусть:

- h - высота экрана;
- w - ширина экрана;
- θ - угол обзора;
- Z_{far} - дальняя граница видимости;
- Z_{near} - ближняя граница видимости.

Тогда M - матрица проекции, вычисляющаяся по формуле 1.12.

$$\begin{bmatrix} \left(\frac{h}{w}\right) \cdot \frac{1}{\tan(\frac{\theta}{2})} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan(\frac{\theta}{2})} & 0 & 0 \\ 0 & 0 & \frac{Z_{far}}{Z_{far}-Z_{near}} & 0 \\ 0 & 0 & \frac{-Z_{far} \cdot Z_{near}}{Z_{far}-Z_{near}} & 0 \end{bmatrix} \quad (1.12)$$

1.15 Описание требований к разрабатываемому программному обеспечению

1.15.1 Описание входных и выходных данных

Входные и выходные данные делятся на две категории. Первая — входные и выходные данные трёхмерного редактора, вторая — входные и выходные данные алгоритма распознавания трёхмерного объекта.

Входные данные трёхмерного редактора включают в себя параметры объектов сцены, вводимые пользователем при помощи интерфейса, адрес файла, содержащего трёхмерный объект, который необходимо загрузить в пространство сцены.

Выходные данные трёхмерного редактора включают в себя параметры объектов сцены, отображающиеся при их изменении, адрес файла, в который сохранён двумерный снимок сцены, адрес файла, в который сохранён трёхмерный объект, находящийся на сцене.

Входные данные алгоритма определения наиболее вероятного соответствия объекта, изображённого на двумерном снимке, с одним из объектов из списка стандартных тел включают в себя изображение, на котором изображён трёхмерный объект.

Выходные данные алгоритма определения наиболее вероятного соответствия объекта, изображённого на двумерном снимке, с одним из объектов из списка стандартных тел включают в себя трёхмерный объект, который был распознанс поданного на вход двумерного изображения.

1.15.2 Описание функциональных требований к программному обеспечению

Разрабатываемое ПО должно предоставлять следующий функционал:

- загрузка полигональных трёхмерных объектов из файлов;
- загрузка трёхмерных объектов на сцену из заготовленного набора примитивов;
- произведение простейших трансформаций (перемещение, масштабирование, вращение) над загруженным на сцену объектом;
- навигация по сцене с помощью перемещения и поворота виртуальной камеры;
- просмотр и изменение свойств объектов в сцене;
- сохранение текущего изображения сцены в файл на диске;
- сохранение просматриваемого трёхмерного объекта в файл;
- определение наиболее вероятного соответствия объекта, изображённого на двумерном снимке, с одним из объектов из списка стандартных тел.

1.16 Вывод из аналитического раздела

В результате проведённого анализа были рассмотрены алгоритмы и методы, позволяющие решить задачу создания программного обеспечения, способного выводить на экран изображение трёхмерного объекта и создавать трёхмерный объект на основе введённого в программу изображения. Были формализованы объекты сцены, включающие в себя источник света, трёхмерный объект и камеру. Был проведён анализ виртуальной камеры и на основе этого анализа сделан вывод о том, что для реализации виртуальной камеры будет использоваться перспективная проекция. Был проведён анализ способов представления поверхности в результате которого сделан вывод о том, что для реализации представления поверхности трёхмерной модели наиболее подходящим является поверхностный способ описания модели, представленный как полигональная сетка, задающаяся списком треугольных полигонов. Был проведён анализ алгоритмов удаления невидимых линий и поверхностей в результате которого сделан вывод о том, что для реализации алгоритма удаления невидимых линий и поверхностей наиболее подходит алгоритм z-буфера. Был проведён анализ методов закрашивания в результате которого сделан вывод о том, что для реализации метода закрашивания наиболее подходит метод простой диффузной закрашки с использованием закона Ламберта. Был проведён анализ алгоритмов машинного обучения для определения наиболее вероятного соответствия объекта, изображённого на двумерном снимке, с одним из объектов из списка стандартных тел в результате которого сделан вывод о том, что для данной задачи наиболее подходит свёрточная нейронная сеть. Были описаны трёхмерные преобразования и преобразования сцены в пространство виртуальной камеры. Были описаны требования к разрабатываемому приложению.

2 Конструкторский раздел

В данном разделе описана структура программного обеспечения, используемые структуры данных, а также приведены схемы алгоритмов, используемых при отрисовке изображения и определения наиболее вероятного соответствия объекта, изображённого на двумерном снимке, с одним из объектов из списка стандартных тел.

2.1 Описание структуры программного обеспечения

В данном подразделе описывается структура разрабатываемого программного обеспечения.

2.1.1 Общая структура программного обеспечения

Программное обеспечение состоит из двух доменов — прикладного, отвечающего за процессы визуализации сцены и распознавания трёхмерного объекта, и интерфейсного, отвечающего за взаимодействие с пользователем

2.1.2 Диаграмма классов

В приложении А на рисунке А.1 приведена диаграмма классов реализуемого программного обеспечения.

2.2 Описание используемых алгоритмов

В данном подразделе рассматриваются основные алгоритмы, использующиеся при решении задачи.

2.2.1 Общий алгоритм отрисовки

Рассмотрим общий алгоритм отрисовки объекта на сцене.

1. Задать экран, на котором должно быть отображено изображение.
2. Задать объект, который должен быть отображён на экране.
3. Инициализировать матрицу просмотра.
4. Заполнить матрицу просмотра значениями, соответствующими параметрам камеры.
5. Инициализировать z-буфер с размерами, соответствующими размерам экрана.
6. Заполнить элементы z-буфера машинной бесконечностью.
7. Инициализировать вектор растеризируемых треугольных полигонов.
8. Для каждого полигона выполнить пункты 8.1—8.8.
 - 8.1. Очистить вектор растеризируемых треугольных полигонов.
 - 8.2. Рассчитать скалярное произведение нормализованного вектора направления направленного источника света с нормалью полигона.
 - 8.3. Если скалярное произведение нормализованного вектора направления направленного источника света с нормалью полигона меньше или равно 0, то полигон не закрашивается и происходит переход на пункт . Иначе выполнить пункты 8.3.1—8.3.2.
 - 8.3.1. Инициализировать цвет полигона значением равным скалярному произведению нормализованного вектора направления направленного источника света с нормалью полигона, умноженному на 255 и на интенсивность источника освещения.
 - 8.3.2. Если цвет полигона больше 255, то установить величину цвета полигона равной 255.
 - 8.4. Инициализировать спроецированный полигон и просматриваемый полигон.
 - 8.5. Перевести полигон в пространство камеры путём умножения векторов вершин полигона на матрицу просмотра.
 - 8.6. Произвести отсечение невидимой части полигона относительно ближней плоскости пространства отсечения.
 - 8.7. Для каждого из отсечённых частей полигона выполнить пункты 8.7.1—8.7.2.

- 8.7.1. Выполнить проецирование полигона на экран.
- 8.7.2. Добавить спроецированный полигон в вектор растеризируемых полигонов.
- 8.8. Для каждого полигона из векторов спроецированных полигонов выполнить пункты 8.8.1—8.8.5.
 - 8.8.1. Масштабировать координаты полигона для соответствия размерам экрана.
 - 8.8.2. Инициализировать список отсечённых полигонов.
 - 8.8.3. Добавить в конец списка отсечённых полигонов рассматриваемый полигон.
 - 8.8.4. Выполнить отсечение полигона относительно плоскостей, ограничивающих пирамиду видимости и результат записать в список отсечённых полигонов.
 - 8.8.5. Для каждого полигона из списка отсечённых полигонов выполнить закраску с учётом z-буфера.
- 9. Обновить экран для вывода полученного изображения.

2.2.2 Алгоритм отсекаания невидимой части треугольного полигона

Алгоритм отсекаания невидимой части треугольного полигона используется в случае, если полигон не находится в пирамиде видимости либо находится внутри её частично. Если не производить отсекание невидимой части, то из-за того, что одна или несколько точек полигона вышли за пределы пирамиды видимости, при расчёте просматриваемого полигона значения будут уходить в сторону бесконечности, что в свою очередь приводит к ощутимой задержке в скорости работы программы, доходящей до полной остановки или аварийного отказа.

Рассмотрим алгоритм отсекаания невидимой части треугольного полигона.

1. Задать точку на плоскости, относительно которой проводится отсечение.
2. Задать нормаль к плоскости, относительно которой проводится отсечение.
3. Задать полигон, подлежащий отсечению.

4. Задать первую часть отсекаемого полигона.
5. Задать вторую часть отсекаемого полигона.
6. Задать нормаль к полигону.
7. Инициализировать вектор вершин, находящихся внутри пирамиды видимости.
8. Инициализировать вектор вершин, находящихся вне пирамиды видимости.
9. Инициализировать количество точек на внутренней стороне плоскости.
10. Инициализировать количество точек на внешней стороне плоскости.
11. Для каждой вершины полигона выполнить пункты 11.1—11.2.
 - 11.1. Вычислить расстояние от вершины до плоскости, относительно которой проводится отсечение.
 - 11.2. Если расстояние больше или равно нулю, то добавить вершину в вектор вершин, находящихся внутри пирамиды видимости. Иначе добавить вершину в вектор вершин, находящихся вне пирамиды видимости.
12. Если количество вершин внутри пирамиды видимости равно нулю, вернуть ноль. Иначе продолжить.
13. Если количество вершин внутри пирамиды видимости равно трём, то выполнить пункты 13.1—13.3. Иначе перейти на пункт 14.
 - 13.1. Скопировать значения полигона, подлежащего отсечению, в поля первого отсекаемого полигона.
 - 13.2. Если скалярное произведение нормали полигона, подлежащего отсечению на нормаль первого отсекаемого полигона меньше нуля, то изменить порядок вершин полигона, подлежащего отсечению так, чтобы удовлетворять общему порядку обхода вершин.
 - 13.3. Вернуть один.
14. Если количество вершин внутри пирамиды видимости равно одному и количество вершин вне пирамиды видимости равно двум, то выполнить пункты 14.1—14.3. Иначе перейти на пункт 15.
 - 14.1. Разделить полигон плоскостью отсечения и часть, находящуюся внутри пирамиды видимости занести в первую часть отсекаемого полигона.
 - 14.2. Если скалярное произведение нормали полигона, подлежащего

отсечению на нормаль первого отсекаемого полигона меньше нуля, то изменить порядок вершин отсечённого полигона, чтобы удовлетворять общему порядку обхода вершин.

14.3. Вернуть один.

15. Если количество вершин внутри пирамиды видимости равно двум и количество вершин вне пирамиды видимости равно одному, то выполнить пункты 15.1—15.5. Иначе перейти на пункт 16.

15.1. Разделить полигон плоскостью отсечения.

15.2. Четырёхугольник, находящийся внутри пирамиды видимости, разделить на два треугольных полигона и занести их в первую часть отсекаемого полигона и вторую часть отсекаемого полигона.

15.3. Если скалярное произведение нормали полигона, подлежащего отсечению на нормаль первого отсекаемого полигона меньше нуля, то изменить порядок вершин отсечённого полигона, чтобы удовлетворять общему порядку обхода вершин.

15.4. Если скалярное произведение нормали полигона, подлежащего отсечению на нормаль второго отсекаемого полигона меньше нуля, то изменить порядок вершин отсечённого полигона, чтобы удовлетворять общему порядку обхода вершин.

15.5. Вернуть два.

16. Вернуть ноль.

2.2.3 Алгоритм z-буфера

Рассмотрим алгоритм z-буфера.

1. Задать вершины треугольного полигона.
2. Задать цвет полигона.
3. Задать z-буфер.
4. Всем элементам буфера кадра присвоить фоновое значение.
5. Инициализировать буфер кадра максимальными значениями глубины.
6. Выполнить растровую развёртку треугольного полигона:
 - 6.1. Для каждого пикселя растрированного полигона вычислить его глубину относительно камеры.
 - 6.2. Сравнить глубину пикселя со значением, которое хранится по координатам этого пикселя в z-буфере. Если данное значение

меньше, чем вычисленное, то оно заменяется значением вычисленного и цвет буфера приравнивается цвету пикселя.

7. Обновить изображение, выводимое на экран в соответствии с буфером кадра.

2.3 Описание архитектуры используемой свёрточной нейросети

Используемая в работе нейросеть состоит из тридцати одного слоя. Состоящего из пяти блоков свёртки и одного блока скрытых слоёв. Блоки свёртки имеют увеличивающуюся глубину слоя, от 16 в первом блоке до 128 в последнем, и уменьшающийся размер ядра свёртки, от 9 до 3. Данная структура позволяет нейросети в первых слоях свёртки распознавать более глобальные признаки и постепенно переходить к распознаванию локальных признаков. Результат последнего блока свёртки трансформируется в одномерный вектор и передаётся на вход скрытым слоям. Результатом работы модели является индекс трёхмерного объекта, наиболее подходящего объекту, изображённому на двумерном изображении.

В приложении Б на рисунке Б.1 приведена схема архитектуры разработанной нейросети.

2.3.1 Алгоритм обучения свёрточной нейросети

Рассмотрим алгоритм обучения свёрточной нейросети.

1. Задать набор изображений для обучения.
2. Преобразовать каждое изображение из набора так, чтобы оно подходило по формату к входному слою модели.
3. Задать количество эпох обучения и параметры оптимизатора модели.
4. Скомпилировать модель.
5. Обучить модель на предоставленных данных.
6. Сохранить результат в файл модели.

2.3.2 Алгоритм определения трёхмерного объекта, наиболее вероятно являющимся объектом изображённым на снимке

Рассмотрим алгоритм определения трёхмерного объекта, наиболее вероятно являющимся объектом изображённым на снимке.

1. Задать снимок.
2. Изменить формат снимка на градации серого.
3. Обрезать изображение по минимальному размеру до квадратного относительно центра.
4. Изменить размер изображения на 256 на 256 пикселей.
5. Загрузить модель нейросети из файла.
6. Для каждого из трёхмерных объектов из списка стандартных объектов вычислить вероятность того, что данный объект изображён на поданном на вход изображении.
7. Вернуть индекс трёхмерного объекта, вероятность которого является максимальной из всех рассчитанных.

2.4 Описание используемых структур данных

В данном подразделе рассматриваются структуры данных, необходимые для решения задачи.

2.4.1 Трёхмерный вектор

Данная структура данных используется при задании положения точки в трёхмерном пространстве. Задаётся вещественными значениями координат точки x , y и z .

2.4.2 Треугольный полигон

Данная структура данных используется при задании трёхмерного объекта, позволяя описать поверхность как совокупность треугольных полигонов. Выбор треугольного полигона обусловлен тем, что три точки

в пространстве однозначно задают плоскость. Задаётся как список из трёх трёхмерных векторов, описывающих положения вершин треугольного полигона.

2.4.3 Матрица

Данная структура данных используется для хранения матриц коэффициентов и расчёта просматриваемой части трёхмерного объекта и проекции трёхмерного объекта на экран. Задаётся как двумерный массив вещественных чисел.

2.4.4 Трёхмерный объект

Данная структура данных используется для описания трёхмерных объектов в пространстве сцены. Описывает основные параметры трёхмерных объектов, включающие в себя положение объекта в пространстве сцены, цвет закрашки объекта и описание поверхности объекта.

2.4.5 Камера

Данная структура данных описывает виртуальную камеру. Используется при синтезе изображения сцены. Описывает ориентацию камеры в пространстве и матрицу просмотра, соответствующую этой камере.

2.4.6 Освещение

Данная структура данных используется для задания источника освещения. Описывает направление распространения, цвет и интенсивность освещения.

2.4.7 Сцена

Данная структура данных используется для задания трёхмерной сцены. Описывает множество находящихся на сцене трёхмерных объектов, камеру и источник освещения.

2.5 Вывод из конструкторского раздела

В данном разделе была рассмотрена структура разрабатываемого программного обеспечения, представлена диаграмма классов и описаны основные алгоритмы, использующиеся в разрабатываемом программном обеспечении. Были определены классы эквивалентности и тесты для каждого из описанных алгоритмов, и описаны используемые структуры данных.

3 Технологический раздел

В данном разделе описываются подробности реализации описанных выше алгоритмов, обосновывается выбор языка программирования и среды разработки, описывают

3.1 Выбор языка программирования и среды разработки

Для написания трёхмерного редактора был выбран язык программирования C++ по следующим причинам:

- C++ имеет статическую типизацию и является компилируемым языком программирования;
- C++ позволяет работать с памятью напрямую и контролировать все вызовы к менеджеру памяти;
- данный язык является объектно-ориентированным, однако позволяет писать программы в структурном стиле.

В качестве среды разработки трёхмерного редактора была выбрана среда QT Creator по следующим причинам:

- данная среда является свободно распространяемым программным обеспечением;
- в данной среде разработки реализовано множество инструментов, которые позволят существенно облегчить процесс написания и отладки кода;
- данная среда предоставляет собственные библиотеки для создания интерфейса для приложения, что существенно упрощает его создание.

Для написания алгоритма распознавания трёхмерного объекта был выбран язык программирования Python 3 по следующим причинам:

- данный язык является объектно-ориентированным;

- для данного языка существует множество библиотек машинного обучения, использование которых позволит существенно сократить время разработки алгоритма.

В качестве среды разработки алгоритма распознавания трёхмерного объекта была выбрана среда Visual Studio Code по следующим причинам:

- данная среда поддерживает разработку на языке Python 3;
- данная среда имеет встроенный интерпретатор языка Python 3;
- данная среда позволяет установить расширения, дающие возможность производить отладку программ, написанных на языке Python 3.

3.2 Листинги реализаций основных алгоритмов

В данном подразделе приводятся листинги реализации основных алгоритмов, использовавшихся при создании программного обеспечения.

3.2.1 Реализация общего алгоритма отрисовки

Ниже в листинге 3.1 представлена реализация общего алгоритма отрисовки.

Листинг 3.1: Реализация общего алгоритма отрисовки

```

1 void SceneManager::render(CanvasLabel &RenderScreen, Mesh &mesh)
2 {
3     Matrix viewMatrix;
4     viewMatrix = mainCamera.formViewMatrix();
5
6     std::vector<std::vector<float>> z_buff(RenderScreen.size().height(),
7         std::vector<float>(RenderScreen.size().width()));
8
9     for (auto &row : z_buff)
10     {
11         for (auto &elem : row)
12         {
13             elem = std::numeric_limits<float>::max();
14         }
15     }
16
17     vector<Triangle> vecTrianglesToRaster;
18
19     for (auto &tri : mesh.tris)
20     {

```

```

20     vecTrianglesToRaster.clear();
21
22     float dp =
23         directionalLight.direction.normalized().scalar(tri.getNormal());
24
25     QColor color;
26     if (dp <= 0)
27     {
28         color.setRgb(0, 0, 0);
29     }
30     else
31     {
32         int colorVal = dp * 255 * directionalLight.power;
33         if (colorVal > 255)
34             colorVal = 255;
35         color.setRgb(colorVal, colorVal, colorVal);
36     }
37     tri.color = color;
38
39     Triangle triangleProjected, triangleViewed;
40
41     tri.vertices[0].vdot(viewMatrix, triangleViewed.vertices[0]);
42     tri.vertices[1].vdot(viewMatrix, triangleViewed.vertices[1]);
43     tri.vertices[2].vdot(viewMatrix, triangleViewed.vertices[2]);
44
45     int clipped_num;
46     Triangle clipped[2];
47     clipped_num = triangleViewed.ClipAgainstPlane({0.0f, 0.0f,
48         renderParams_.fNear}, {0.0f, 0.0f, 1.0f}, triangleViewed,
49         clipped[0], clipped[1], triangleViewed.getNormal());
50
51     for (int n = 0; n < clipped_num; n++)
52     {
53         clipped[n].vertices[0].vdot(projMatrix_,
54             triangleProjected.vertices[0]);
55         clipped[n].vertices[1].vdot(projMatrix_,
56             triangleProjected.vertices[1]);
57         clipped[n].vertices[2].vdot(projMatrix_,
58             triangleProjected.vertices[2]);
59
60         triangleProjected.vertices[0].z_ = clipped[n].vertices[0].z_;
61         triangleProjected.vertices[1].z_ = clipped[n].vertices[1].z_;
62         triangleProjected.vertices[2].z_ = clipped[n].vertices[2].z_;
63
64         vecTrianglesToRaster.push_back(triangleProjected);
65     }
66
67     for (auto &triToRaster : vecTrianglesToRaster)

```

```

62     {
63         triToRaster.vertices[0].x_ *= 0.1 * RenderScreen.size().width();
64         triToRaster.vertices[0].y_ *= 0.1 * RenderScreen.size().height();
65
66         triToRaster.vertices[1].x_ *= 0.1 * RenderScreen.size().width();
67         triToRaster.vertices[1].y_ *= 0.1 * RenderScreen.size().height();
68
69         triToRaster.vertices[2].x_ *= 0.1 * RenderScreen.size().width();
70         triToRaster.vertices[2].y_ *= 0.1 * RenderScreen.size().height();
71
72         Triangle clipped[2];
73         Triangle test_temp;
74         list<Triangle> listTriangles;
75
76         listTriangles.push_back(triToRaster);
77         int nNewTriangles = 1;
78
79
80         for (int p = 0; p < 4; p++)
81         {
82             int nTrisToAdd = 0;
83             while (nNewTriangles > 0)
84             {
85                 Triangle test = listTriangles.front();
86                 listTriangles.pop_front();
87                 nNewTriangles--;
88                 switch (p)
89                 {
90                     case 0:
91                         nTrisToAdd = test_temp.ClipAgainstPlane({ 0.0f,
92                             (float)RenderScreen.height()/2 - 1, 0.0f }, { 0.0f,
93                             -1.0f, 0.0f }, test, clipped[0], clipped[1],
94                             test.getNormal());
95                         break;
96                     case 1:
97                         nTrisToAdd = test_temp.ClipAgainstPlane({ 0.0f,
98                             -(float)RenderScreen.height()/2 + 1, 0.0f }, { 0.0f,
99                             1.0f, 0.0f }, test, clipped[0], clipped[1],
100                             test.getNormal());
101                         break;
102                     case 2:
103                         nTrisToAdd = test_temp.ClipAgainstPlane({
104                             (float)RenderScreen.width()/2 - 1, 0.0f, 0.0f }, {
105                             -1.0f, 0.0f, 0.0f }, test, clipped[0], clipped[1],
106                             test.getNormal());
107                         break;
108                     case 3:

```

```

100         nTrisToAdd = test_temp.ClipAgainstPlane({
            -(float)RenderScreen.width()/2 + 1, 0.0f, 0.0f }, {
            1.0f, 0.0f, 0.0f }, test, clipped[0], clipped[1],
            test.getNormal());
101         break;
102     }
103
104     for (int w = 0; w < nTrisToAdd; w++)
105         listTriangles.push_back(clipped[w]);
106
107     }
108     nNewTriangles = listTriangles.size();
109 }
110
111
112 for (auto &tris : listTriangles)
113 {
114     RenderScreen.FillTriangleWithZBuffer(
115         tris.vertices[0],
116         tris.vertices[1],
117         tris.vertices[2],
118         tri.color,
119         z_buff
120     );
121 }
122 }
123 }
124
125 RenderScreen.updateCanvas();
126 }

```

3.2.2 Реализация алгоритма отсекаания невидимой части треугольного полигона

Ниже в листинге 3.2 представлена реализация алгоритма отсекаания невидимой части треугольного полигона.

Листинг 3.2: Реализация общего алгоритма отрисовки

```

1 int Triangle::ClipAgainstPlane(Vector3 plane_point, Vector3 plane_normal,
   Triangle &in_triangle, Triangle &out_triangle_1, Triangle &out_triangle_2,
   Vector3 tri_normal)
2 {
3     plane_normal = plane_normal.normalized();
4     tri_normal.normalize();
5     auto dist = [&](Vector3 p)

```

```

6   {
7       return (plane_normal.x_ * p.x_ + plane_normal.y_ * p.y_ +
8           plane_normal.z_ * p.z_ - plane_normal.scalar(plane_point));
9   };
10
11   Vector3* inside_points[3];
12   int nInsidePointCount = 0;
13   Vector3* outside_points[3];
14   int nOutsidePointCount = 0;
15
16   float d0 = dist(in_triangle.vertices[0]);
17   float d1 = dist(in_triangle.vertices[1]);
18   float d2 = dist(in_triangle.vertices[2]);
19
20   if (d0 >= 0)
21       inside_points[nInsidePointCount++] = &in_triangle.vertices[0];
22   else
23       outside_points[nOutsidePointCount++] = &in_triangle.vertices[0];
24
25   if (d1 >= 0)
26       inside_points[nInsidePointCount++] = &in_triangle.vertices[1];
27   else
28       outside_points[nOutsidePointCount++] = &in_triangle.vertices[1];
29
30   if (d2 >= 0)
31       inside_points[nInsidePointCount++] = &in_triangle.vertices[2];
32   else
33       outside_points[nOutsidePointCount++] = &in_triangle.vertices[2];
34
35   if (nInsidePointCount == 0)
36       return 0;
37
38   if (nInsidePointCount == 3)
39   {
40       out_triangle_1 = in_triangle;
41
42       if (out_triangle_1.getNormal().normalized().scalar(tri_normal) < 0)
43       {
44           Vector3 temp = out_triangle_1.vertices[2];
45           out_triangle_1.vertices[2] = out_triangle_1.vertices[1];
46           out_triangle_1.vertices[1] = temp;
47       }
48       return 1;
49   }
50
51   if (nInsidePointCount == 1 && nOutsidePointCount == 2)
52   {
53       out_triangle_1 = in_triangle;

```

```

53     out_triangle_1.vertices[0] = *inside_points[0];
54     out_triangle_1.vertices[1] = getPlaneIntersection(plane_point,
55         plane_normal, *inside_points[0], *outside_points[0]);
56     out_triangle_1.vertices[2] = getPlaneIntersection(plane_point,
57         plane_normal, *inside_points[0], *outside_points[1]);
58
59     if (out_triangle_1.getNormal().normalized().scalar(tri_normal) < 0)
60     {
61         Vector3 temp = out_triangle_1.vertices[2];
62         out_triangle_1.vertices[2] = out_triangle_1.vertices[1];
63         out_triangle_1.vertices[1] = temp;
64     }
65
66     return 1;
67 }
68
69 if (nInsidePointCount == 2 && nOutsidePointCount == 1)
70 {
71     out_triangle_1 = in_triangle;
72     out_triangle_2 = in_triangle;
73
74     out_triangle_1.vertices[0] = *inside_points[0];
75     out_triangle_1.vertices[1] = *inside_points[1];
76     out_triangle_1.vertices[2] = getPlaneIntersection(plane_point,
77         plane_normal, *inside_points[0], *outside_points[0]);
78
79     if (out_triangle_1.getNormal().normalized().scalar(tri_normal) < 0)
80     {
81         Vector3 temp = out_triangle_1.vertices[2];
82         out_triangle_1.vertices[2] = out_triangle_1.vertices[1];
83         out_triangle_1.vertices[1] = temp;
84     }
85
86     out_triangle_2.vertices[0] = *inside_points[1];
87     out_triangle_2.vertices[1] = out_triangle_1.vertices[2];
88     if (abs(out_triangle_2.vertices[1].x_ - out_triangle_2.vertices[0].x_)
89         < 0.0001 &&\
90         abs(out_triangle_2.vertices[1].y_ - out_triangle_2.vertices[0].y_)
91         < 0.0001 &&\
92         abs(out_triangle_2.vertices[1].z_ - out_triangle_2.vertices[0].z_)
93         < 0.0001)
94     {
95         out_triangle_2.vertices[1] = out_triangle_1.vertices[1];
96         out_triangle_2.vertices[2] = getPlaneIntersection(plane_point,
97             plane_normal, *inside_points[1], *outside_points[0]);
98
99         if (out_triangle_2.getNormal().normalized().scalar(tri_normal) < 0)
100         {
101             Vector3 temp = out_triangle_2.vertices[2];

```

```

94         out_triangle_2.vertices[2] = out_triangle_2.vertices[1];
95         out_triangle_2.vertices[1] = temp;
96     }
97
98     return 2;
99 }
100 return 0;
101 }

```

3.2.3 Реализация алгоритма z-буфера

Ниже в листинге 3.3 представлена реализация алгоритма z-буфера.

Листинг 3.3: Реализация общего алгоритма отрисовки

```

1 void CanvasLabel::FillTriangleWithZBuffer(Vector3 p_1, Vector3 p_2, Vector3
   p_3, QColor color, std::vector<std::vector<float>> &z_buff)
2 {
3     int x_max = get_max(p_1.x_, p_2.x_, p_3.x_);
4     int y_max = get_max(p_1.y_, p_2.y_, p_3.y_);
5
6     int x_min = get_min(p_1.x_, p_2.x_, p_3.x_);
7     int y_min = get_min(p_1.y_, p_2.y_, p_3.y_);
8
9     Vector3 temp_p_1;
10    Vector3 temp_p_2;
11    Vector3 temp_p_3;
12
13    temp_p_1.y_ = y_max;
14    if (int(temp_p_1.y_) == int(p_1.y_))
15        temp_p_1 = p_1;
16    if (int(temp_p_1.y_) == int(p_2.y_))
17        temp_p_1 = p_2;
18    if (int(temp_p_1.y_) == int(p_3.y_))
19        temp_p_1 = p_3;
20
21    temp_p_3.y_ = y_min;
22    if (int(temp_p_3.y_) == int(p_1.y_) && int(temp_p_1.y_) != int(p_1.y_))
23        temp_p_3 = p_1;
24    if (int(temp_p_3.y_) == int(p_2.y_) && int(temp_p_1.y_) != int(p_2.y_))
25        temp_p_3 = p_2;
26    if (int(temp_p_3.y_) == int(p_3.y_) && int(temp_p_1.y_) != int(p_3.y_))
27        temp_p_3 = p_3;
28
29    if ((temp_p_1 == p_1 && temp_p_3 == p_2) || (temp_p_1 == p_2 && temp_p_3 ==
   p_1))
30        temp_p_2 = p_3;

```



```

31     if ((temp_p_1 == p_1 && temp_p_3 == p_3) || (temp_p_1 == p_3 && temp_p_3 ==
        p_1))
32         temp_p_2 = p_2;
33     if ((temp_p_1 == p_2 && temp_p_3 == p_3) || (temp_p_1 == p_3 && temp_p_3 ==
        p_2))
34         temp_p_2 = p_1;
35
36     for (int y = y_min; y < y_max; y++)
37     {
38         for (int x = x_min; x < x_max; x++)
39         {
40             if (abs(x) < this->size().width()/2 && abs(y) <
                this->size().height()/2)
41             {
42                 float e21 = TestPoint(x, y, p_2.x_, p_2.y_, p_1.x_, p_1.y_);
43                 float e32 = TestPoint(x, y, p_3.x_, p_3.y_, p_2.x_, p_2.y_);
44                 float e13 = TestPoint(x, y, p_1.x_, p_1.y_, p_3.x_, p_3.y_);
45
46                 if (e21 >= 0.0f && e32 >= 0.0f && e13 >= 0.0f)
47                 {
48                     if (y == y_max || y == y_min)
49                     {
50                         if ((y == int(temp_p_2.y_) && (y == int(temp_p_1.y_))))
51                         {
52                             float z = temp_p_1.z_ + (temp_p_2.z_ - temp_p_1.z_)
                                * ((x - temp_p_1.x_)/(temp_p_2.x_ -
                                    temp_p_1.x_));
53                             if (z < z_buff[this->size().height()/2 - y][x +
                                    this->size().width()/2])
54                             {
55                                 setPixelColor(x, y, color);
56                                 z_buff[this->size().height()/2 - y][x +
                                    this->size().width()/2] = z;
57                             }
58                             //setPixelColor(x, y, color);
59                         }
60                         else if ((y == int(temp_p_2.y_) && (y ==
                            int(temp_p_3.y_))))
61                         {
62                             float z = temp_p_3.z_ + (temp_p_2.z_ - temp_p_3.z_)
                                * ((x - temp_p_3.x_)/(temp_p_2.x_ -
                                    temp_p_3.x_));
63                             if (z < z_buff[this->size().height()/2 - y][x +
                                    this->size().width()/2])
64                             {
65                                 setPixelColor(x, y, color);
66                                 z_buff[this->size().height()/2 - y][x +
                                    this->size().width()/2] = z;

```

```

67         }
68         //setPixelColor(x, y, color);
69     }
70     else if (y == int(temp_p_1.y_))
71     {
72         float z = temp_p_1.z_;
73         if (z < z_buff[this->size().height()/2 - y][x +
74             this->size().width()/2])
75         {
76             setPixelColor(x, y, color);
77             z_buff[this->size().height()/2 - y][x +
78                 this->size().width()/2] = z;
79         }
80         //setPixelColor(x, y, color);
81     }
82     else if (y == int(temp_p_3.y_))
83     {
84         float z = temp_p_3.z_;
85         if (z < z_buff[this->size().height()/2 - y][x +
86             this->size().width()/2])
87         {
88             setPixelColor(x, y, color);
89             z_buff[this->size().height()/2 - y][x +
90                 this->size().width()/2] = z;
91         }
92         //setPixelColor(x, y, color);
93     }
94 }
95 else if (y >= int(temp_p_2.y_))
96 {
97     int x_a = temp_p_1.x_ + (temp_p_2.x_ - temp_p_1.x_)*((y
98         - temp_p_1.y_)/(temp_p_2.y_ - temp_p_1.y_));
99     int x_b = temp_p_1.x_ + (temp_p_3.x_ - temp_p_1.x_)*((y
100         - temp_p_1.y_)/(temp_p_3.y_ - temp_p_1.y_));
101     float z_a = temp_p_1.z_ + (temp_p_2.z_ -
102         temp_p_1.z_)*((y - temp_p_1.y_)/(temp_p_2.y_ -
103         temp_p_1.y_));
104     float z_b = temp_p_1.z_ + (temp_p_3.z_ -
105         temp_p_1.z_)*((y - temp_p_1.y_)/(temp_p_3.y_ -
106         temp_p_1.y_));
107
108     if (x_a != x_b)
109     {
110         float z = z_a + (z_b - z_a)*((x - x_a)/(x_b - x_a));
111
112         if (z < z_buff[this->size().height()/2 - y][x +
113             this->size().width()/2])
114         {

```

```

104         setPixelColor(x, y, color);
105         z_buff[this->size().height()/2 - y][x +
            this->size().width()/2] = z;
106     }
107 }
108 else
109 {
110     if (z_a < z_buff[this->size().height()/2 - y][x +
        this->size().width()/2])
111     {
112         setPixelColor(x, y, color);
113         z_buff[this->size().height()/2 - y][x +
            this->size().width()/2] = z_a;
114     }
115 }
116 }
117 else if (y < int(temp_p_2.y_))
118 {
119     int x_a = temp_p_2.x_ + (temp_p_3.x_ - temp_p_2.x_)*((y
        - temp_p_2.y_)/(temp_p_3.y_ - temp_p_2.y_));
120     int x_b = temp_p_1.x_ + (temp_p_3.x_ - temp_p_1.x_)*((y
        - temp_p_1.y_)/(temp_p_3.y_ - temp_p_1.y_));
121     float z_a = temp_p_2.z_ + (temp_p_3.z_ -
        temp_p_2.z_)*((y - temp_p_2.y_)/(temp_p_3.y_ -
        temp_p_2.y_));
122     float z_b = temp_p_1.z_ + (temp_p_3.z_ -
        temp_p_1.z_)*((y - temp_p_1.y_)/(temp_p_3.y_ -
        temp_p_1.y_));
123
124     if (x_a != x_b)
125     {
126         float z = z_a + (z_b - z_a)*((x - x_a)/(x_b - x_a));
127
128         if (z < z_buff[this->size().height()/2 - y][x +
            this->size().width()/2])
129         {
130             setPixelColor(x, y, color);
131             z_buff[this->size().height()/2 - y][x +
                this->size().width()/2] = z;
132         }
133         //setPixelColor(x, y, color);
134     }
135     else
136     {
137         if (z_a < z_buff[this->size().height()/2 - y][x +
            this->size().width()/2])
138         {
139             setPixelColor(x, y, color);

```

```

140         z_buff[this->size().height()/2 - y][x +
141             this->size().width()/2] = z_a;
142     }
143 }
144 }
145 }
146 }
147 }
148 }

```

3.2.4 Реализация алгоритма обучения свёрточной нейросети

Ниже в листинге 3.4 представлена реализация алгоритма обучения свёрточной нейросети.

Листинг 3.4: Реализация алгоритма обучения свёрточной нейросети

```

1 import tensorflow as tf
2 from tensorflow import keras
3 from keras import layers
4 import numpy as np
5 import cv2
6
7 train_materials_path =
8     "D:/Storage/Andrew/BMSTU/BMSTU/CW/Work/CourseProject/TrainMaterials/"
9
10 import os, os.path
11
12 x_train = []
13 y_train = []
14
15 #size of dataset
16 dataset_size = len(os.listdir(train_materials_path))
17 classes = 8
18 image_size = (256, 256)
19 image_shape = (256, 256, 1)
20 batch_size = 128
21
22 import os
23
24 clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
25 count = 0
26
27 for name in os.listdir(train_materials_path):
28     fpath_img = train_materials_path + name
29     image = cv2.imread(fpath_img, cv2.IMREAD_GRAYSCALE)

```

```

29
30 min_dim = min(image.shape)
31 w = min_dim
32 h = min_dim
33 center = image.shape
34 x = center[1]/2 - w/2
35 y = center[0]/2 - h/2
36
37 crop_img = image[int(y):int(y+h), int(x):int(x+w)]
38
39 img_clh = clahe.apply(crop_img)
40 new_img = cv2.resize(img_clh, image_size)
41
42 x_train.append(new_img)
43
44 index = int(name.split('_')[1])
45 res_vec = [0] * classes
46 res_vec[index] = 1
47
48 y_train.append(res_vec)
49 count += 1
50 if count \% 100 == 0:
51     os.system('cls')
52     print(count, ':', dataset_size)
53
54 x_train = np.array(x_train)
55 y_train = np.array(y_train)
56
57 from keras.layers.pooling import MaxPooling2D
58 from keras import Sequential
59 from keras.layers import Conv2D, Flatten, Dense, MaxPool2D, AveragePooling2D,
    BatchNormalization
60
61 model = Sequential()
62
63 model.add(Conv2D(16, kernel_size=(9,9), strides=(1,1), padding='same',
    input_shape=image_shape))
64 model.add(Conv2D(16, kernel_size=(9,9), strides=(1,1), padding='same'))
65 model.add(Conv2D(16, kernel_size=(9,9), strides=(1,1), padding='same'))
66 model.add(Conv2D(16, kernel_size=(9,9), strides=(1,1), padding='same'))
67 model.add(MaxPooling2D(pool_size=(2,2)))
68
69 model.add(Conv2D(32, kernel_size=(7,7), strides=(1,1), padding='same'))
70 model.add(Conv2D(32, kernel_size=(7,7), strides=(1,1), padding='same'))
71 model.add(Conv2D(32, kernel_size=(7,7), strides=(1,1), padding='same'))
72 model.add(Conv2D(32, kernel_size=(7,7), strides=(1,1), padding='same'))
73 model.add(MaxPooling2D(pool_size=(2,2)))
74

```

```

75 model.add(Conv2D(64, kernel_size=(5,5), strides=(1,1), padding='same'))
76 model.add(Conv2D(64, kernel_size=(5,5), strides=(1,1), padding='same'))
77 model.add(Conv2D(64, kernel_size=(5,5), strides=(1,1), padding='same'))
78 model.add(Conv2D(64, kernel_size=(5,5), strides=(1,1), padding='same'))
79 model.add(MaxPooling2D(pool_size=(2,2)))
80
81 model.add(Conv2D(128, kernel_size=(3,3), strides=(1,1), padding='same'))
82 model.add(Conv2D(128, kernel_size=(3,3), strides=(1,1), padding='same'))
83 model.add(Conv2D(128, kernel_size=(3,3), strides=(1,1), padding='same'))
84 model.add(Conv2D(128, kernel_size=(3,3), strides=(1,1), padding='same'))
85 model.add(MaxPooling2D(pool_size=(2,2)))
86
87 model.add(Conv2D(128, kernel_size=(3,3), strides=(1,1), padding='same'))
88 model.add(Conv2D(128, kernel_size=(3,3), strides=(1,1), padding='same'))
89 model.add(Conv2D(128, kernel_size=(3,3), strides=(1,1), padding='same'))
90 model.add(Conv2D(128, kernel_size=(3,3), strides=(1,1), padding='same'))
91 model.add(MaxPooling2D(pool_size=(2,2)))
92
93 model.add(Flatten())
94
95 model.add(Dense(256, activation='relu'))
96 model.add(Dense(128, activation='relu'))
97 model.add(Dense(64, activation='relu'))
98 model.add(Dense(32, activation='relu'))
99 model.add(Dense(classes, activation='softmax'))
100
101 from keras import losses
102 from keras import metrics
103
104 epochs = 20
105
106 opt = tf.keras.optimizers.Adam(learning_rate=0.001, decay=1e-3 / 200)
107
108 callbacks = [
109     keras.callbacks.ModelCheckpoint("save_model_at_{epoch}.h5"),
110 ]
111 model.compile(loss=losses.categorical_crossentropy, optimizer=opt,
112               metrics=metrics.Accuracy())
113
114 history = model.fit(
115     x_train, y_train, epochs=epochs, callbacks=callbacks, validation_split=0.2,
116 )
117 model.save('new_model.h5')

```

3.2.5 Реализация алгоритма определения трёхмерного объекта, наиболее вероятно являющимся объектом изображённым на снимке

Ниже в листинге 3.5 представлена реализация общего алгоритма отрисовки.

Листинг 3.5: Реализация алгоритма определения трёхмерного объекта наиболее вероятно являющимся объектом изображённым на снимке

```
1 def predict_model(fpath_img: str) -> int:
2     import cv2
3     import numpy as np
4     import keras
5     import tensorflow as tf
6     from keras import layers
7
8     try:
9         image_size = (256, 256)
10
11         clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
12         image = cv2.imread(fpath_img, cv2.IMREAD_GRAYSCALE)
13
14         max_index = 0
15         count = 0
16         max_val = 0
17
18         min_dim = min(image.shape)
19         w = min_dim
20         h = min_dim
21         center = image.shape
22         x = center[1]/2 - w/2
23         y = center[0]/2 - h/2
24
25         crop_img = image[int(y):int(y+h), int(x):int(x+w)]
26
27         img_clh = clahe.apply(crop_img)
28         new_img = cv2.resize(img_clh, image_size)
29
30         from keras.models import load_model
31         import os
32         dir_path = os.path.dirname(os.path.realpath(__file__))
33         model = load_model(dir_path + "\model.h5")
34
35         img_array = tf.expand_dims(new_img, 0)
36         predictions = model.predict(img_array)
```

```

37
38     for pred in predictions[0]:
39         if max_val < pred:
40             max_val = pred
41             max_index = count
42             count += 1
43
44     return max_index
45 except:
46     return -1

```

3.3 Сведения о модулях программы

Разработанное программное обеспечение состоит из следующих модулей:

- `main.cpp` — основной модуль программы, содержащий точку входа в программу, последовательность команд, инициализирующих интерфейс и сцену.
- `baseobject` — модуль программы, содержащий описание класса базового объекта и реализацию методов этого класса.
- `camera` — модуль программы, содержащий описание класса камеры и реализацию методов этого класса.
- `canvaslabel` — модуль программы, содержащий описание класса холста и реализацию методов этого класса.
- `directionallight` — модуль программы, содержащий описание класса направленного источника освещения.
- `mainwindow` — модуль программы, содержащий описание основного класса интерфейса и реализацию методов этого класса.
- `matrix` — модуль программы, содержащий описание класса матрицы и реализацию методов этого класса.
- `mesh` — модуль программы, содержащий описание класса трёхмерного объекта, задающегося множеством треугольных полигонов, и реализацию методов этого класса.
- `meshloader` — модуль программы, содержащий описание класса, отвечающего за загрузку из файла и сохранение в файл трёхмерных объектов, и реализацию методов этого класса.
- `scenemanager` — модуль программы, содержащий структуру,

описывающую параметры синтеза изображения, описание класса менеджера сцены и реализацию методов этого класса.

- `triangle` — модуль программы, содержащий описание класса треугольного полигона и реализацию методов этого класса.
- `vector3` — модуль программы, содержащий описание класса трёхмерного вектора и реализацию методов этого класса.
- `predict.py` — модуль программы, содержащий алгоритм обучения свёрточной нейросети.
- `train.py` — модуль программы, содержащий алгоритм определения трёхмерного объекта, наиболее вероятно являющимся объектом изображённым на снимке.

3.3.1 Описание данных, используемых при обучении свёрточной нейросети

Для обучения свёрточной перонной сети, использовавшейся в проекте, использовался набор снимков, сгенерированных при помощи реализованного редактора. Для этого для каждой из моделей из списка стандартных моделей было создано восемь серий изображений. Каждая из серий содержит в себе снимки изображения, повернутого на угол от нуля до ста восьмидесяти по каждой из осей OX , OY и OZ . При этом при сохранении снимка в названии файла был указан номер снимка и индекс объекта из списка, который являлся основой для снимка. Всего было сгенерировано 11520 снимков для 8 различных моделей, при для каждой модели было сгенерировано 1440 снимка.

3.4 Информация, необходимая для сборки и запуска разработанного программного обеспечения

3.4.1 Сборка программного обеспечения

В исходном виде разработанное программное обеспечение представляет из себя проект среды разработки Qt Creator, поэтмоу для сборки проекта

необходимо выполнить следующие действия:

1. открыть проект при помощи среды разработки Qt Creator;
2. изменить параметры сборки проекта, установив комплект сборки Desktop Qt 5.15.2 MinGW 64-bit;
3. выбрать вид сборки, в зависимости от цели это может быть сборка для выпуска, сборка для отладки и сборка для профилирования, при успешной сборке проекта в папке проекта будет создана директория `build-CourseProject-Desktop_Qt_5_15_2_MinGW_64_bit` (окончание названия данной папки будет варьироваться в зависимости от вида сборки, `-debug` — сборка для отладки, `-release` — сборка для выпуска, `-profile` — сборка для профилирования);
4. в созданную дерикторию с собранным проектом в папку с объектными файлами необходимо скопировать папку `PresetModels` и содержимое папки `CNN`, после чего программное обеспечение будет полностью собрано и готово к работе.

3.4.2 Запуск программного обеспечения

Для запуска программного обеспечения необходимо запустить исполняемый файл `CourseProject.exe`, находящийся в папке с объектными файлами в директории, которая содержит файлы собранного проекта.

3.5 Интерфейс разработанного программного обеспечения

В данном подразделе описываются элементы интерфейса разработанного программного обеспечения.

3.5.1 Общие элементы интерфейса программы

В данном подразделе описываются общие элементы интерфейса.

Холст

Холст позволяет выводить изображение, синтезированное в процессе работы программы, на экран. Ниже на рисунке 3.1 приведена реализация холста в графическом интерфейсе редактора

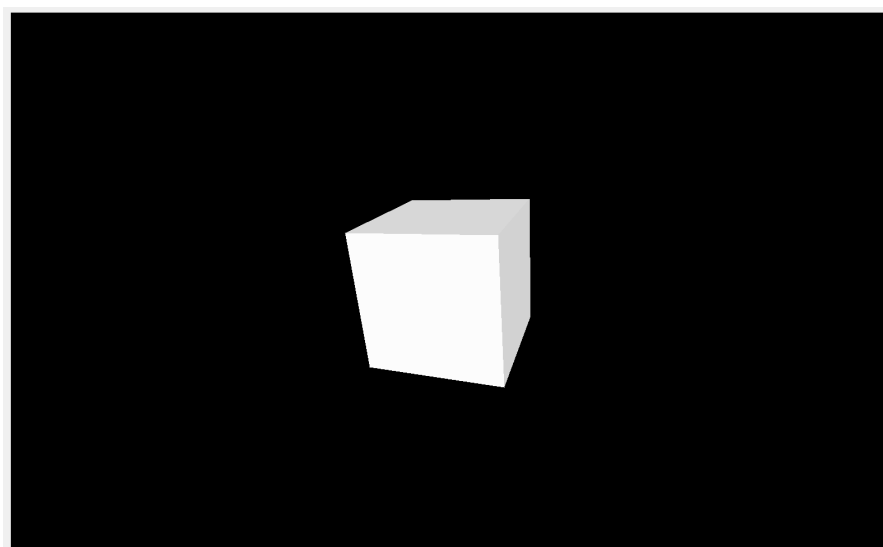


Рис. 3.1: Холст

Интерфейс перемещения между группами элементов интерфейса

Интерфейс перемещения между группами интерфейса позволяет пользователю просматривать различные группы элементов интерфейса, относящиеся к определённым элементам редактора. Реализован в виде двух кнопок, позволяющих просматривать предыдущую и следующую группу элементов интерфейса. Ниже на рисунке 3.2 приведена реализация интерфейса перемещения в графическом интерфейсе редактора.

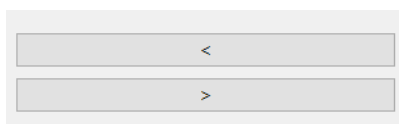


Рис. 3.2: Интерфейс перемещения между группами интерфейса

3.5.2 Интерфейс параметров модели

Интерфейс параметров модели позволяет просматривать и изменять параметры модели, сохранять модель и загружать модели либо из объектного файла, либо из списка стандартных моделей. Ниже на рисунке 3.4 приведена

реализация интерфейса параметров модели в графическом интерфейсе редактора.

Параметры модели

Положение

x	y	z
0,00	0,00	0,00

Поворот

x	y	z
-22,35	20,63	0,00

Масштаб

x	y	z
1,00	1,00	1,00

Применить

Сбросить преобразования

Сохранение модели

Сохранить модель

Загрузка модели

Загрузить модель из файла

Куб

Загрузить встроенную модель

Рис. 3.3: Интерфейс параметров модели

3.5.3 Интерфейс параметров камеры

Интерфейс параметров камеры позволяет просматривать и изменять параметры виртуальной камеры. Ниже на рисунке 3.4 приведена реализация интерфейса параметров камеры в графическом интерфейсе редактора.

Параметры камеры

Положение

x	y	z
0,00	0,00	-6,00

Поворот

x	y	z
0,00	0,00	0,00

Применить

Сбросить преобразования

Рис. 3.4: Интерфейс параметров виртуальной камеры

3.5.4 Интерфейс параметров освещения

Интерфейс параметров освещения позволяет просматривать и изменять параметры источника освещения. Ниже на рисунке 3.5 приведена реализация интерфейса параметров освещения в графическом интерфейсе редактора.

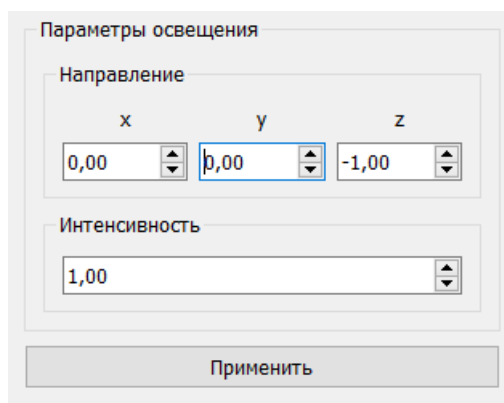


Рис. 3.5: Интерфейс параметров освещения

3.5.5 Интерфейс синтеза изображения

Интерфейс параметров освещения позволяет создавать снимок сцены и сохранять его в виде файла формата png. Ниже на рисунке 3.6 приведена реализация интерфейса синтеза изображения в графическом интерфейсе редактора.

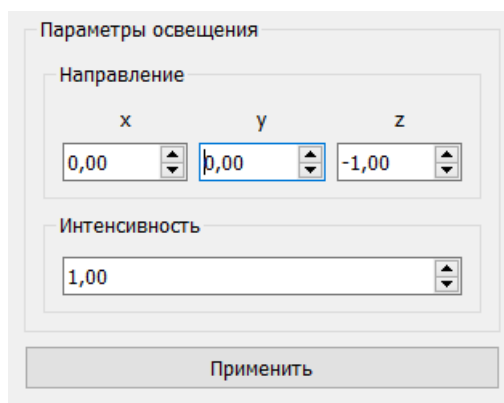


Рис. 3.6: Интерфейс синтеза изображения

3.5.6 Интерфейс определения наиболее вероятного соответствия объекта, изображённого на двумерном снимке

Интерфейс сопределения наиболее вероятного соответствия объекта, изображённого на двумерном снимке загружать файл с изображением трёхмерного объекта для определения модели из списка стандартных моделей, которая наиболее вероятно изображена на загруженном изображении. Ниже на рисунке 3.7 приведена реализация данного интерфейса в графическом интерфейсе редактора.

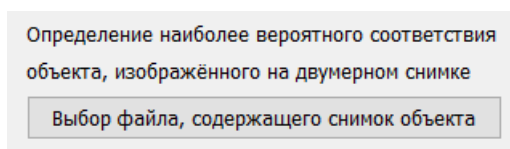


Рис. 3.7: Интерфейс определения наиболее вероятного соответствия объекта, изображённого на двумерном снимке

3.6 Вывод из технологического раздела

В данном разделе был рассмотрен выбор языка программирования и среды разработки для проекта. Были представлены листинги реализаций основных алгоритмов, использованных при решении задачи. Были представлены сведения о модулях программы и информация, необходимая для сборки и запуска разработанного программного обеспечения. Был описан интерфейс разработанного программного обеспечения.

4 Экспериментальный раздел

В данном разделе будут поставлены эксперименты по оценке зависимости скорости работы программы от количества полигонов в просматриваемой модели и по оценке эффективности работы разработанного алгоритма определения трёхмерного объекта, наиболее вероятно являющимся объектом изображённым на снимке.

4.1 Технические характеристики

Ниже приведены технические характеристики компьютера, на котором проводились эксперименты.

- Операционная система - Windows 10, 64-bit;
- Оперативная память - 16 GiB;
- Процессор - Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz 2.59 GHz, 6 ядер, 12 потоков.

4.2 Исследование зависимости скорости работы программы от количества полигонов в загружаемой модели

В данном подразделе описывается эксперимент по оценке зависимости скорости работы алгоритма отрисовки от количества полигонов в просматриваемой модели.

4.2.1 Цель эксперимента по исследованию зависимости скорости работы алгоритма отрисовки от количества полигонов в просматриваемой модели

Целью эксперимента является исследование зависимости скорости работы алгоритма отрисовки от количества полигонов в просматриваемой модели, и определение свойств этой зависимости.

4.2.2 Описание эксперимента по исследованию зависимости скорости работы алгоритма отрисовки от количества полигонов в просматриваемой модели

В данном эксперименте в редактор загружался ряд трёхмерных объектов, обладающих различным количеством полигонов. Всего было исследовано 8 трёхмерных объектов, имеющих различное количество полигонов от 20 до 1280.

4.2.3 Результат эксперимента по исследованию зависимости скорости работы алгоритма отрисовки от количества полигонов в просматриваемой модели

Результаты эксперимента представлены ниже в таблице 4.2 и на графике 4.1.

Таблица 4.1: Время отрисовки в зависимости от количества полигонов

количество полигонов	время отрисовки (нс)
20	80491760.0
40	76552860.0
80	72199800.0
160	64196280.0
320	61207200.0
720	52398480.0
1280	48407940.0

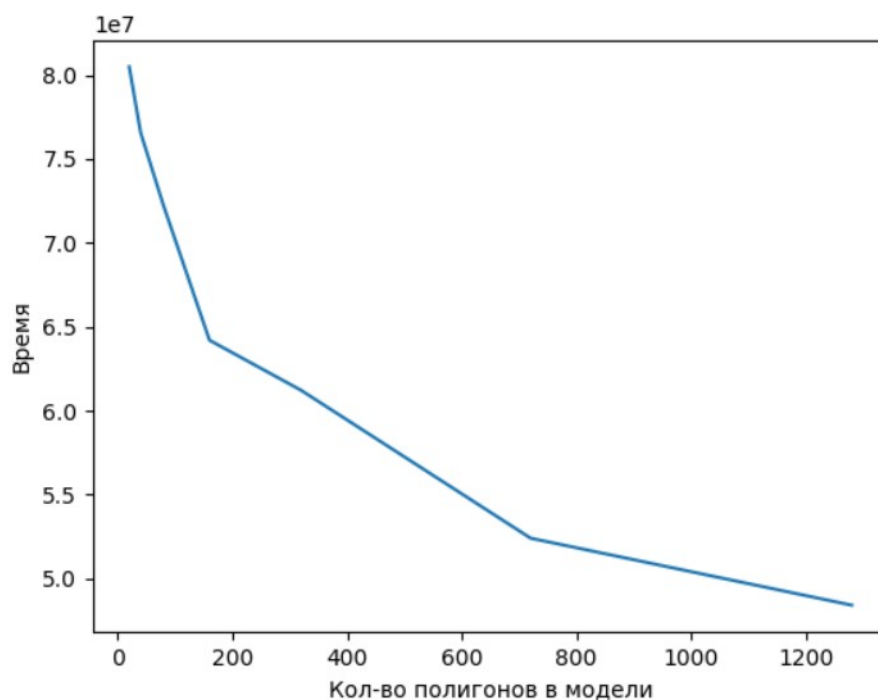


Рис. 4.1: Визуализация результатов эксперимента

Из полученных результатов можно сделать вывод о том, что при увеличении количества полигонов время отрисовки изображения уменьшается.

4.3 Исследование эффективности работы разработанного алгоритма определения трёхмерного объекта, наиболее вероятно являющимся объектом изображённым на снимке

В данном разделе описывается эксперимент по оценке эффективности работы разработанного алгоритма определения трёхмерного объекта, наиболее вероятно являющимся объектом изображённым на снимке.

4.3.1 Цель эксперимента по исследованию эффективности работы разработанного алгоритма определения трёхмерного объекта, наиболее вероятно являющимся объектом изображённым на снимке

Целью эксперимента является оценка эффективности работы разработанного алгоритма определения трёхмерного объекта, наиболее вероятно являющимся объектом изображённым на снимке.

4.3.2 Описание эксперимента по исследованию эффективности работы разработанного алгоритма определения трёхмерного объекта, наиболее вероятно являющимся объектом изображённым на снимке

В данном эксперименте создаётся набор из 5 снимков для каждого из объектов из списка стандартных объектов, после чего данные снимки подаются на вход алгоритму. Полученный объект сравнивается с тем, который служил основой для снимка. Результат высчитывается как процентное соотношение количества корректно определённых результатов к количеству некорректно определённых результатов.

4.3.3 Результат эксперимента по исследованию эффективности работы разработанного алгоритма определения трёхмерного объекта, наиболее вероятно являющимся объектом изображённым на снимке

Результаты эксперимента представлены ниже в таблице

Таблица 4.2: Соотношение количества корректно определённых результатов к количеству некорректно определённых результатов для каждого из рассматриваемых трёхмерных объектов

трёхмерный объект	корректно распознанные из пяти
куб	0
тетраэдр	5
пирамида	4
сфера	5
изосфера	5
тор	5
цилиндр	5
конус	5

Из полученных результатов можно сделать вывод о том, что алгоритм корректно распознаёт объекты, имеющие отличающуюся внешнюю форму, такие как тор, конус, цилиндр, изосферу и сферу. Однако если объекты имеют похожую геометрию, то алгоритм не во всех случаях может определить корректную модель, что происходит в случае куба, пирамиды и тетраэдра.

4.4 Вывод из экспериментального раздела

В данном разделе были поставлены эксперименты по оценке зависимости скорости работы алгоритма отрисовки от количества полигонов в просматриваемой модели и по оценке эффективности работы разработанного алгоритма определения трёхмерного объекта, наиболее вероятно являющимся объектом изображённым на снимке.

В результате эксперимента по оценке зависимости скорости работы алгоритма отрисовки от количества полигонов в просматриваемой модели было выявлено, что скорость работы алгоритма отрисовки зависит от количества полигонов таким образом, что чем больше полигонов, тем быстрее работает алгоритм отрисовки, достигая разницы более чем в 1.6 раз.

В результате эксперимента по оценке эффективности работы разработанного алгоритма определения трёхмерного объекта, наиболее вероятно являющимся объектом изображённым на снимке было выявлено, что

алгоритм корректно распознаёт объекты, имеющие отличающуюся внешнюю форму, такие как тор, конус, цилиндр, изосферу и сферу. Однако если объекты имеют похожую геометрию, то алгоритм не во всех случаях может определить корректную модель, что происходит в случае куба, пирамиды и тетраэдра.

Заключение

В рамках курсового проекта реализована программа, позволяющая загружать, просматривать, редактировать масштаб, позицию и вращение трёхмерной модели, синтезировать изображение трёхмерной модели, и определять наиболее вероятное соответствие объекта, изображённого на двумерном снимке, с одним из объектов из списка стандартных тел.

В ходе выполнения проекта были выполнены следующие задачи:

- исследованы подходы к синтезу изображений;
- выполнено сравнение существующих решений;
- описаны используемые при разработке ПО алгоритмы;
- определены средства программной реализации;
- реализованы алгоритмы отрисовки сцены;
- реализованы алгоритмы генерации изображения;
- реализован алгоритм позволяющий определять наиболее вероятное соответствие объекта, изображённого на двумерном снимке, с одним из объектов из списка стандартных тел;
- исследована производительность алгоритма отрисовки для различных объектов;
- исследована эффективность работы алгоритма восстановления трёхмерного объекта из его двумерного снимка.

В результате проведённого исследования оценки зависимости скорости работы алгоритма отрисовки от количества полигонов в просматриваемой модели было выявлено, что скорость работы алгоритма отрисовки зависит от количества полигонов таким образом, что чем больше полигонов, тем быстрее работает алгоритм отрисовки, достигая разницы более чем в 1.6 раз.

В результате проведённого исследования оценки эффективности работы разработанного алгоритма определения трёхмерного объекта, наиболее вероятно являющимся объектом изображённым на снимке было выявлено, что

алгоритм корректно распознаёт объекты, имеющие отличающуюся внешнюю форму, такие как тор, конус, цилиндр, изосферу и сферу. Однако если объекты имеют похожую геометрию, то алгоритм не во всех случаях может определить корректную модель, что происходит в случае куба, пирамиды и тетраэдра.

Список использованных источников

- [1] Педди, Джон. История визуальной магии в компьютерах : как создаются красивые изображения в САПР, 3D, VR и AR. — Лондон, 2013. — С. 25.
- [2] Роджерс Д., Адамс Дж. Математические основы машинной графики. — Москва, Мир, 2001. — С. 604.
- [3] Шилов А. В., Лесковец И. В. Компьютерная графика. — Могилев, Государственное учреждение высшего профессионального образования «Белорусско-Российский университет», 2014 — С. 38.
- [4] Шишкин Е. В., Боресков А. В., Зайцев А. А., Компьютерная графика. — Москва, ”ДИАЛОГ-МИФИ 1993. — С. 135.
- [5] Шишкин Е. В., Боресков А. В., Компьютерная графика. — Москва, ”ДИАЛОГ-МИФИ 1996. — С. 290.
- [6] Valencia-Garcia, Rafael, Technologies and Innovation: Second International Conference. — Guayaquil, Ecuador, 2016. — С. 146.
- [7] Дэви С., Арно М., Мохамед А., Основы Data Science и Big Data. Python и наука о данных. — Санкт-Петербург, 2017. — С. 336
- [8] Нейронная сеть // Большая российская энциклопедия : [в 35 т.] / гл. ред. Ю. С. Осипов. — Москва, Большая российская энциклопедия, 2004-2017.
- [9] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard and L. D. Jackel: Backpropagation Applied to Handwritten Zip Code Recognition, Neural Computation — MIT, 1989 — С. 541-551.

Приложение А содержит диаграмму классов разрабатываемого программного обеспечения.



Приложение Б

Приложение Б содержит схему архитектуры разработанной нейронной сети.

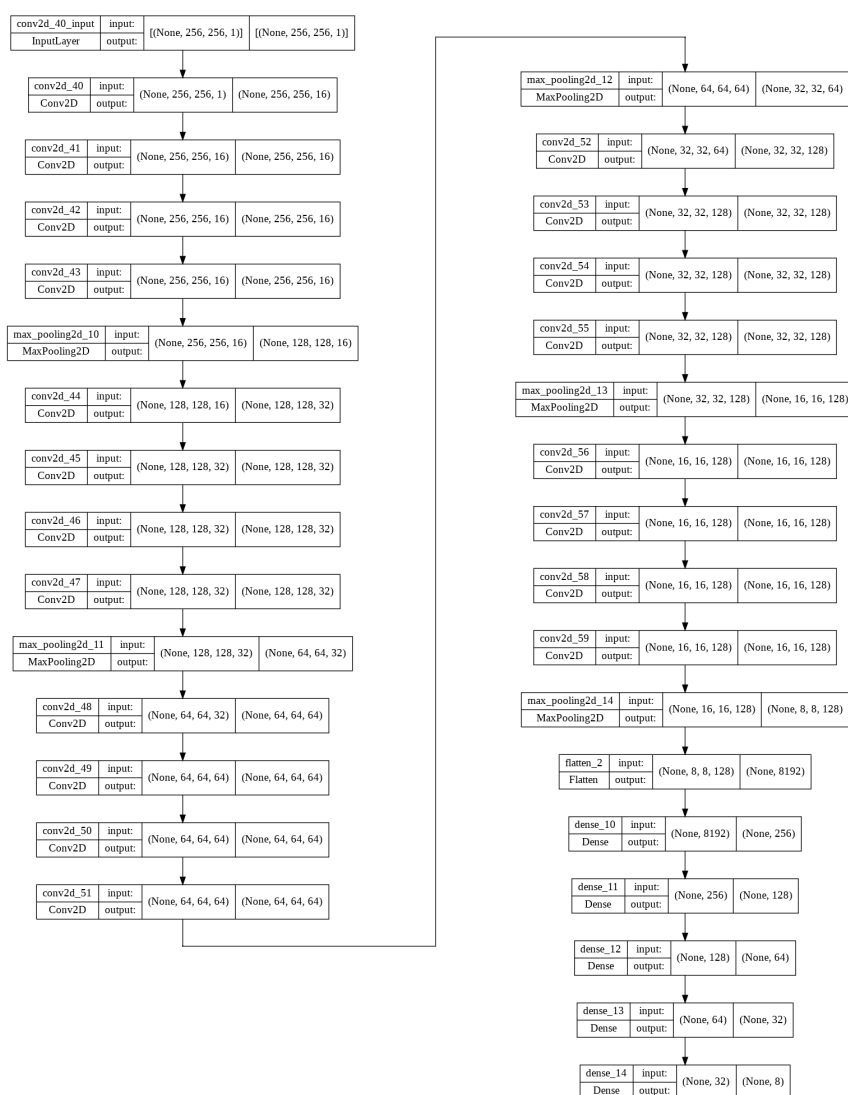


Рис. Б.1: Схема архитектуры разработанной нейронной сети