**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 тел»*

Студент ИУ7-55Б _____ Бугаенко А. П.
(Группа) (Подпись, дата) (Фамилия И. О.)

Руководитель курсовой работы _____ Новик Н. В.
(Подпись, дата) (Фамилия И. О.)

*2022 г.*

# Содержание

# Введение

В современном мире компьютерная графика используется во многих сферах человеческой деятельности, 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, широко используется в различных индустриях, начиная с киноиндустрии и заканчивая разработкой игр и симуляторов.

Теория распознавания образа — раздел информатики и смежных дисциплин, развивающий основы и методы классификации и идентификации предметов, явлений, процессов, сигналов, ситуаций и объектов, которые характеризуются конечным набором некоторых свойств и признаков. Такие задачи решаются довольно часто, например, при переходе или проезде улицы по сигналам светофора. Распознавание цвета загоревшейся лампы светофора и знание правил дорожного движения позволяет принять правильное решение о том, можно или нельзя переходить улицу. Необходимость в таком распознавании возникает в самых разных областях — от военного дела и систем безопасности до оцифровки аналоговых сигналов.

На сегодняшний день проблема распознавания образа приобрела выдающееся значение в условиях информационных перегрузок, когда человек не справляется с линейно-последовательным пониманием поступающих к нему сообщений, в результате чего его мозг переключается на режим одновременности восприятия и мышления, которому свойственно такое распознавание.

Целью данной работы является создание программного обеспечения, которое позволило бы загружать трёхмерную модель, редактировать масштаб, вращение и положение трёхмерной модели, выводить на экран и сохранять изображение трёхмерной модели, и соответствие объекта, изображённого на

двумерном снимке, с одним из объектов из списка тел.

Трёхмерная модель загружается из объектного файла (.obj) в пространство сцены. Модель должна быть задана полигонально, поскольку таким образом снимок подели будет содержать достаточно данных для работы алгоритма нахождения подходящей модели из списка. Пространство сцены представляет из себя трёхмерное пространство, в котором находятся объекты сцены, включающие в себя виртуальную камеру, источники освещения и трёхмерный объект. Виртуальная камера представляет из себя наблюдателя в пространстве сцены, позволяющего получать двумерную проекцию сцены, которая затем отображается на экране.

Для достижения поставленной цели необходимо решить следующие задачи:

- исследовать подходы к синтезу изображений;
- исследовать способы создания двумерного снимка;
- исследовать алгоритмы нахождения трёхмерного объекта по двумерному снимку;
- описать используемые при разработке ПО алгоритмы;
- определить средства программной реализации;
- реализовать алгоритмы отрисовки сцены;
- реализовать алгоритм позволяющий определять наиболее вероятное соответствие объекта, изображённого на двумерном снимке, с одним из объектов из списка тел;
- исследовать зависимость скорости алгоритма отрисовки от количества полигонов в объекте;
- исследовать зависимость скорости алгоритма отрисовки от размера проекции полигона;
- исследовать эффективность работы алгоритма алгоритма определения наиболее вероятного соответствия объекта на изображении объекту из списка тел.

# 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 методов.

## 1.1 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.

## 1.2 Анализ виртуальной камеры

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

Проекция — изображение трёхмерной фигуры на так называемой проекционной плоскости (экране) способом, представляющим собой геометрическую идеализацию оптических механизмов зрения, фотографии, камеры-обскуры.

Существует два основных вида проецирования:

- параллельная проекция;
- перспективная проекция.

Параллельная проекция — это проекция объекта в трехмерном 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.

Для поставленной задачи наилучшим решением будет использовать перспективную проекцию, поскольку она содержит больше информации о положении трёхмерного объекта в пространстве.

## 1.3 Описание способа представления поверхности

Трёхмерная модель в разрабатываемом программном обеспечении загружается из объектного файла, имеющего формат OBJ. Данный формат подразумевает сохранение модели в поверхностном виде с помощью полигональной сетки, поэтому для описания модели также используется поверхностный способ и полигональная сетка. Поскольку информация в файле содержит только описание точек и рёбер, то необходимо определить, каким образом будет представлена полигональная сетка. 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.

Для поставленной задачи наилучшим решением будет использование списка треугольных полигонов, поскольку треугольный полигон однозначно задаёт плоскость в трёхмерно пространстве и не требует дополнительных проходов по списку треугольных полигонов при отрисовке, что увеличивает

скорость работы программы.

## 1.4 Анализ алгоритмов удаления невидимых линий и поверхностей

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 получаются 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 их 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].

Для поставленной задачи наилучшим решением для реализации алгоритма удаления невидимых линий и поверхностей будет использование алгоритма z-буфера, поскольку в данной задаче важна скорость построения изображения для динамической отрисовки трёхмерной сцены программы для просмотра моделей.

## 1.5 Анализ методов закрашивания

В компьютерной графике для расчета освещенности граней объектов применяется цветовая модель RGB. Интенсивность отраженного света точек объектов вычисляют отдельно для каждой их 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 $\theta$ 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.

$$I_d = I_p \cdot k_d \cdot cos\theta. \tag{1.1}$$

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 нормалями, которые сами тоже являются результатами интерполяции вдоль ребер многоугольника между нормалями 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].

Для поставленной задачи наилучшим решением будет метод простой диффузной закраски с использованием закона Ламберта, поскольку программа для просмотра моделей должна работать в режиме реального времени и не требуется рассчёт отражений и бликов при построении изображения модели.

## 1.6 Описание получения изображения трёхмерного объекта

Получение изображения трёхмерного объекта подразумевает преобразование математической пространственной модели в плоскую (растровую) картинку. Как структура данных, изображение на экране представляется матрицей точек, где каждая точка определена, по крайней мере, тремя числами: интенсивностью красного, синего и зелёного цвета. Таким образом, рендеринг преобразует трёхмерную векторную структуру данных в плоскую матрицу пикселей.

Для того, чтобы построить двумерное изображение нам необходимо получить проекцию сцены относительно наблюдателя, находящегося в пространстве сцены. При этом, для того, чтобы преобразовать сцену в пространство камеры (наблюдателя), 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;
- $\theta$ - угол обзора;
- $Z_{far}$ - дальнаяя граница видимости;
- $Z_{near}$ - ближняя граница видимости.

Тогда $M$ - матрица проекции, вычисляющаяся по формуле 1.2.

$$\begin{bmatrix} (\frac{h}{w}) \cdot \frac{1}{tan(\frac{\theta}{2})} & 0 & 0 & 0 \\ 0 & \frac{1}{tan(\frac{\theta}{2})} & 0 & 0 \\ 0 & 0 & \frac{Z_{far}}{Z_{far}-Z_{near}} & 0 \\ 0 & 0 & \frac{-Z_{far} \cdot Z_{near}}{Z_{far}-Z_{near}} & 0 \end{bmatrix} \tag{1.2}$$

После того, как сцена была преобразована в пространство камеры, происходит растрирование каждого треугольного полигона модели. Для пикселей этого полигона устанавливается цветовое значение с учётом цвета полигона и освещённости полигона, которая рассчитывается по закону Ламберта. После чего область на изображении, соответствующая полигону, закрашивается с учётом невидимых объектов, который производится с помощью алгоритма z-буффера.

Для того, чтобы получить снимок модели, сначала создаётся изображение модели. После чего создаётся файл формата JPEG, куда заносится полученная при создании изображения матрица пикселей. Затем снимок сохраняется в директории компьютера, указываемой пользователем при запросе на сохранение снимка.

## 1.7 Анализ алгоритмов определения наиболее вероятного соответствия объекта, изображённого на двумерном снимке, с одним из объектов из списка тел

Распознавание образов — это отнесение исходных данных к определённому классу с помощью выделения существенных признаков,

характеризующих эти данные, из общей массы данных.

Классическая постановка задачи распознавания образов: Дано множество объектов. Относительно них необходимо провести классификацию. Множество представлено подмножествами, которые называются классами. Заданы: информация о классах, описание всего множества и описание информации об объекте, принадлежность которого к определённому классу неизвестна. Требуется по имеющейся информации о классах и описании объекта установить к какому классу относится этот объект.

Существует несколько способов распознавания образов:

- Оптическое распознавание — данный метод подразумевает перебор вида объекта под различными углами, масштабами, смещениями. Тот образ, который менее всего отличается от разпознаваемого, выбирается в качестве результата распознания.
- Исследование контура объекта — данный способ подразнайти контур объекта и исследовать его свойства (связность, наличие углов и т.д.).
- Использование исскуственных нейронных сетей — данный метод требует либо большого количества примеров задачи распознавания (с правильными ответами), либо специальной структуры нейронной сети, учитывающей специфику данной задачи.

Для данной задачи наилучшим решением будет использование исскуственных нейронных сетей, так как с помощью реализуемой программы для просмотра модели возможно генерировать примеры задачи распознавания в неограниченном количестве для каждого объекта. Помимо этого нейросети позволяют легко расширять количество классов, для этого достаточно дообучить эту нейросеть на изображениях, полученных с помощью программы для просмотра моделей для нового объекта.

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].

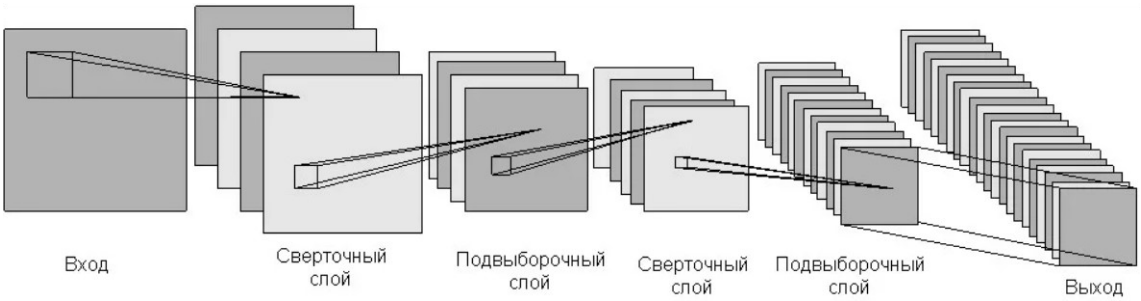


Рис. 1.1: Схема свёрточной нейросети

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].

Для данной задачи наилучшим решением будет использование свёрточной нейронной сети, поскольку свёрточные нейронные сети, позволяют извлекать абстрактные признаки из изображений. Затем эти признаки используются моделью машинного обучения для определения вероятности соответствия входных данных определённому значению выхода нейросети.

## 1.8 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, мастабирование и поворот.

Рассмотрим операцию перемещения. Пусть координаты некоторой точки в трёхмерном пространстве задаются вектором $\overrightarrow{p} = \{p_x, p_y, p_z\}$, а вектор перемещения задаётся вектором $\overrightarrow{d} = \{d_x, d_y, d_z\}$, тогда координаты перемещённой точки в пространстве задаются вектором $\overrightarrow{p'}$, который вычисляется по формуле 1.3.

$$\overrightarrow{p'} = \begin{cases} p'_x = p_x + d_x \\ p'_y = p_y + d_y \\ p'_z = p_z + d_z \end{cases} \tag{1.3}$$

Операция перемещения в матричном виде вычисляется по формуле 1.4.

$$\overrightarrow{p'} = \begin{bmatrix} p_x & p_y & p_z & 0 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ d_x & d_y & d_z & 1 \end{bmatrix} \tag{1.4}$$

Рассмотрим операцию мастабирования. Данная операция происходит относительно точки, называемой центром масштабирования. Пусть в этом случае центр масштабирования - начало координат, координаты некоторой точки в трёхмерном пространстве задаются вектором $\overrightarrow{p} = \{p_x, p_y, p_z\}$, а вектор коэффициентов масштабирования точки задаётся вектором $\overrightarrow{s} = \{s_x, s_y, s_z\}$, тогда координаты масштабированной относительно центра координат точки в пространстве задаются вектором $\overrightarrow{p'}$, который вычисляется по формуле 1.5.

$$\overrightarrow{p'} = \begin{cases} p'_x = p_x \cdot s_x \\ p'_y = p_y \cdot s_y \\ p'_z = p_z \cdot s_z \end{cases} \tag{1.5}$$

Операция масштабирования в матричном виде вычисляется по формуле

1.6.

$$\overrightarrow{p'} = \begin{bmatrix} p_x & p_y & p_z & 0 \end{bmatrix} \times \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.6}$$

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 $\overrightarrow{p} = \{p_x, p_y, p_z\}$, а угол поворота относительно оси равен $phi$. Тогда координаты повернутой относительно осей $OX, OY, OZ$ точки в пространстве задаются векторами $\overrightarrow{p'_{OX}}, \overrightarrow{p'_{OY}}, \overrightarrow{p'_{OZ}}$, которые соответственно вычисляются по формулам 1.7, 1.8, 1.9.

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

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

$$\overrightarrow{p'_{OZ}} = \begin{cases} p'_x = p_x \cdot cos\phi + p_y \cdot sin\phi \\ p'_y = -p_x \cdot sin\phi + p_z \cdot cos\phi \\ p'_z = p_z \end{cases} \tag{1.9}$$

Операции поворота вокруг осей $OX, OY, OZ$ в матричном виде вычисляются по формулам 1.10, 1.11, 1.12.

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

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

$$\overrightarrow{p'_{OZ}} = \begin{bmatrix} p_x & p_y & p_z & 0 \end{bmatrix} \times \begin{bmatrix} cos\phi & -sin\phi & 0 & 0 \\ sin\phi & cos\phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.12}$$

## 1.9 Описание требований к разрабатываемому программному обеспечению

### 1.9.1 Описание входных и выходных данных

Входные и выходные данные делятся на две категории. Первая — входные и выходные данные программы для просмотра модели, вторая — входные и выходные данные алгоритма распознания трёхмерного объекта.

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

Выходные данные программы для просмотра модели 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 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 тел включают в себя трёхмерный объект, который был

распознанс поданного на вход двумерного изображения.

### 1.9.2 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 тел.

## 1.10 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. Были формализованы объекты сцены, включающие в себя источник света, трёхмерный объект и камеру. Для реализации виртуальной камеры была выбрана перспективная проекция. Для реализации представления поверхности трёхмерной модели был выбран поверхностный способ описания модели, представленный как полигональная сетка, задающаяся списком треугольных полигонов. Для реализации алгоритма удаления невидимых линий и поверхностей был выбран алгоритм z-буфера. Для реализации метода закрашивния был выбран метод простой диффузной закраски с использованием закона Ламберта. Для реализации алгоритма, позволяющего определять объект из списка объектов, изображённого на изображении, с

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

## 2.1 Описание структуры программного обеспечения

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

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 в разрабатываемом программном обеспечении.

### 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, умноженому 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 отсекания невидимой части треугольного полигона

Алгоритм отсекания 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 рассчёте просматриваемого полигона значения его полей будут уходить в бесконечноть, что в свою очередь приводит аварийному завершению работы программы.

Рассмотрим алгоритм отсекания 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 пирамины 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

Алгоритм z-буфера приводится в виде схемы в приложении В на рисунках В.1—В.2.

## 2.3 Описание архитектуры используемой свёрточной нейросети

Использующаяся 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 архитектуты 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 объектов вычислить вероятность того, что данный объект изображён на поданном на вход изображении.
7. Вернуть индекс трёхмерного объекта, вероятность которого является максимальной из всех рассчитаных.

## 2.4 Описание используемых структур данных

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

Трёхмерный вектор — эта структура данных использующется

при задании положения точки в трёхмерном пространстве. Задаётся вещественными значениями координат точки $x$, $y$ и $z$.

Треугольный полигон — эта 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.

Матрица — эта стурктура данных используется для хранения матриц коэффициентов и рассчёта просматриваемой части трёхмерного объекта и проекции трёхмерного объекта на экран. Задаётся как двумерный массив вещественных чисел.

Трёхмерный объект — это структура данных используется для описания трёхмерных объектов в пространстве сцены. Описывает основные параметры трёхмерных объектов, включающие в себя положение объекта в пространстве сцены, цвет закраски объекта и описание поверхности объекта.

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

Освещение — эта структура данных используется для задания источника овещения. Описывает направление распространения, цвет и интенсивность освещения.

Сцена — эта 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, обосновывется выбор языка программирования и среды разработки, описывают

## 3.1 Выбор языка программирования и среды разработки

Для написания программы для просмотра моделей 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 программы для просмотра моделей 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 реализацию.

Для реализации программы нахождения наиболее вероятного совпадения был выбран язык программирования Python 3 по следующим причинам:

- данный язык является объектно-ориентированным;

- для данного языка существуем множество хорошо оптимизированных библиотек машинного обучения, использование которых позволит существенно сократить время разработки и обучения модели.

В качестве среды разработки программы определения наиболее вероятного совпадения 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

Реализации основных алгоритмов приводятся в приложении Г в листингах Г.1 — Г.5.

## 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 интрефейс 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, использующихся при обучении свёрточной нейросети

Для обучения свёрточной неронной сети, использовавшейся в проекте, использовался набор снимков, сгенерированных при помощи реализованной программы для просмотра моделей. Для этого для каждой из моделей из списка 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 указан номер снимка и индекс объекта из списка, который являлся основой для снимка. Всего было сгененрировано 11520 снимков для 8 различных моделей, при для каждой модели было сгенерировано 1440 снимка.

Изображения моделей, использовавшихся для генерации данных

приведены в приложении Д на рисунках Д.1—Д.8.

## 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 в среде разработки Qt Creator. Для его сборки 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 позволяет выводить изображение, синтезированное в процессе работы программы, на экран. Ниже на рисунке 3.1 приведена реализация холста в графическом интерфейсе программы для просмотра моделей

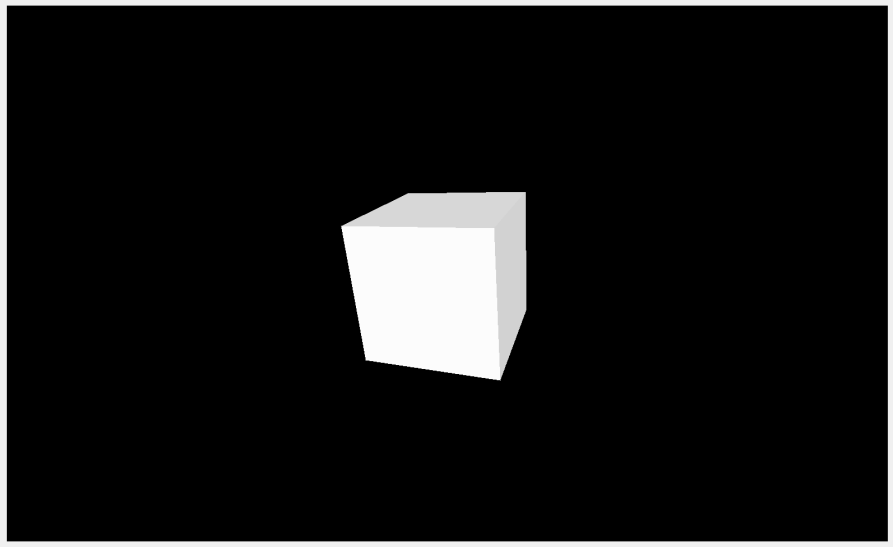

Рис. 3.1: Холст

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

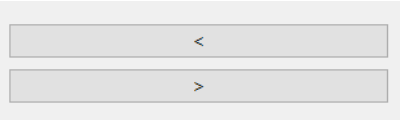


Рис. 3.2: Интерфейс перемещения между группами интерфейса

Интейфейс параметров модели позволяет просматривать и изменять параметры модели, сохранять модель и загружать модели либо из объектного файла, либо из списка моделей. Ниже на рисунке 3.4 приведена реализация интерфейса параметров модели в графическом интерфейсе программы для просмотра моделей.

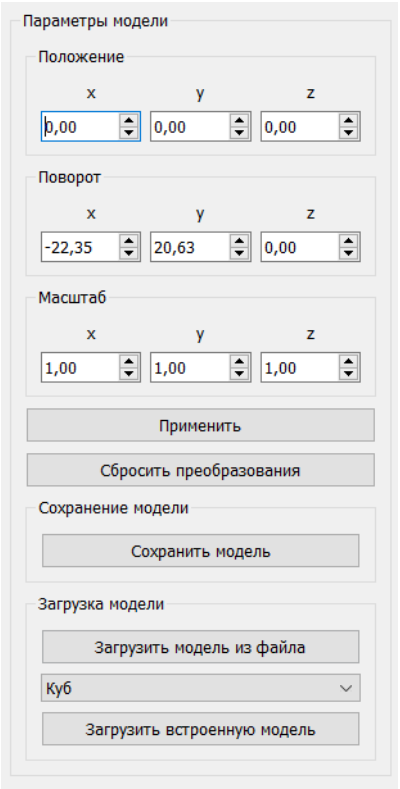


Рис. 3.3: Интерфейс параметров модели

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

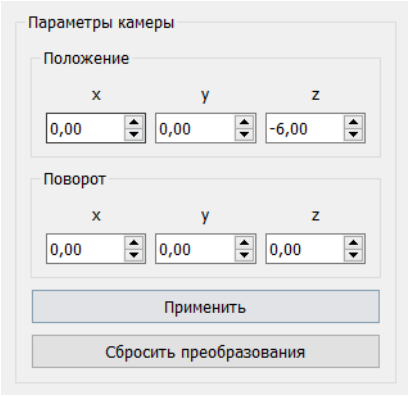


Рис. 3.4: Интерфейс параметров виртуальной камеры

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

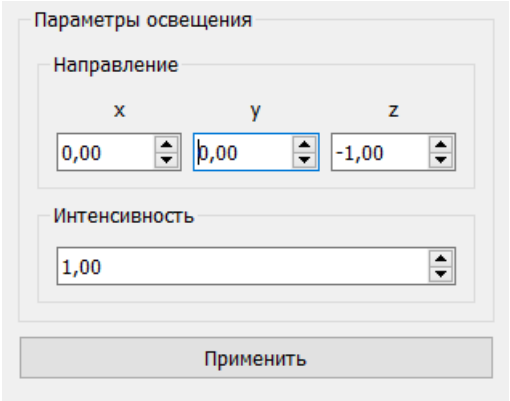


Рис. 3.5: Интерфейс параметров освещения

Интейрфейс параметров освещения позволяет создавать снимок сцены и сохранять его в виде файла формата png. Ниже на рисунке 3.6 приведена реализация интерфейса синтеза изображения графическом интерфейсе программы для просмотра моделей.

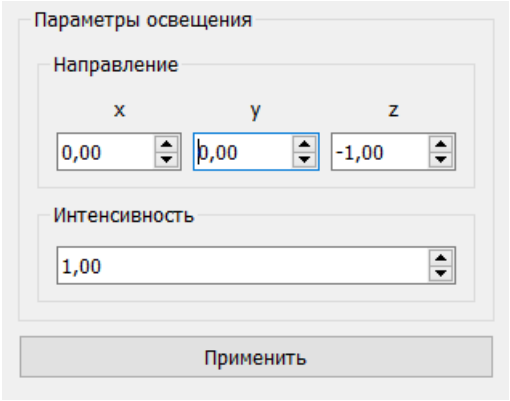


Рис. 3.6: Интерфейс синтеза изображения

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

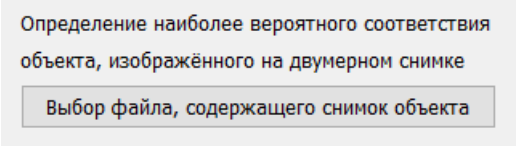


Рис. 3.7: Интерфейс определения наиболее вероятного соответствия объекта, изображённого на двумерном снимке

## 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

Техническихе 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.

## 4.1 Тестирование работы программы

В данном подразделе описывается системное тестирование разработанного программного обеспечения. При загрузке программного обеспечения открывается окно, содержащее интерфейс работы с моделью и экран с изображённым на нём кубом. Вид окна приведён ниже на рисунке 4.1.

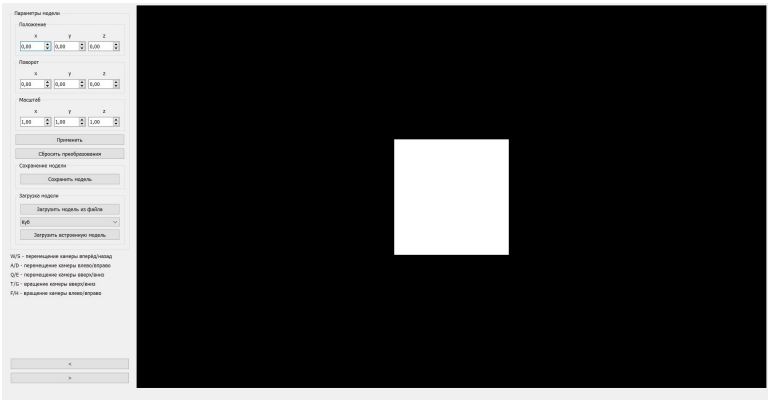

Рис. 4.1: Окно, содержащее интерфейс работы с моделью

Произвёдем тестирование фукнции загрузки файла из модели. Для этого загрузим модель тора из файла. Ниже на рисунках 4.2—4.3 представлены интерфейс загрузки файла и результат загрузки модели в программу.

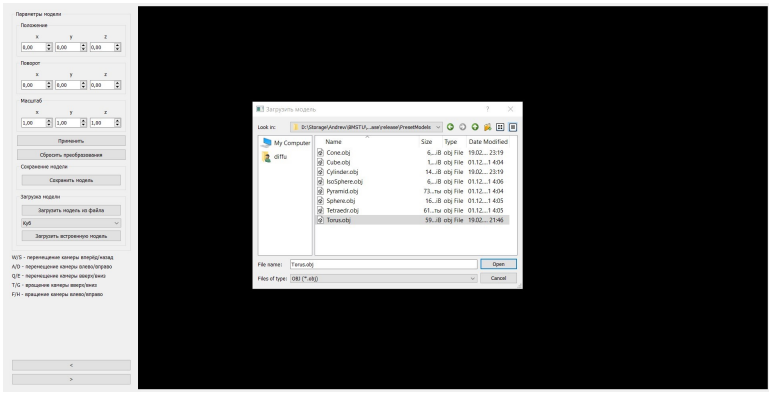

Рис. 4.2: Интерфейс загрузки файла

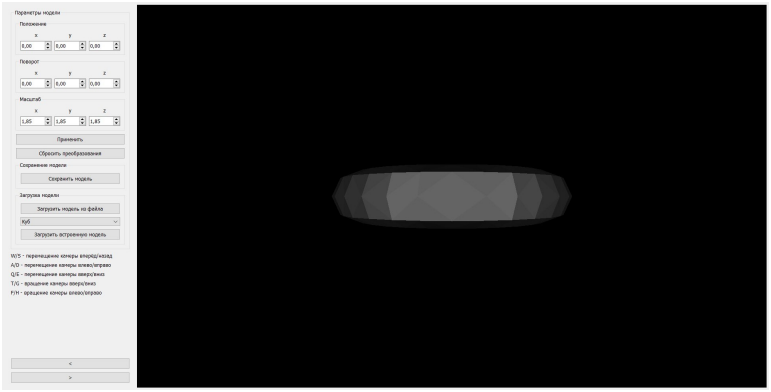

Рис. 4.3: Загруженная на сцену модель тора

Произведём тестирование функции сохранения снимка модели в виде файла с изображением. Ниже на рисунках 4.4—4.8 представлены интерфейс сохранения снимка и полученный снимок.

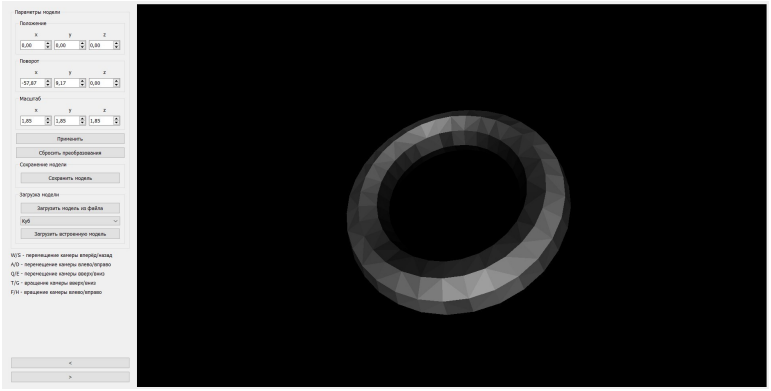

Рис. 4.4: Вид сцены, который будет записан в снимок

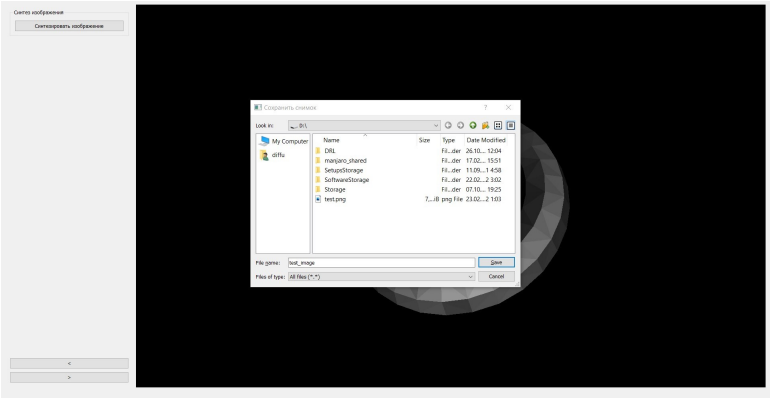

Рис. 4.5: Интерфейс сохранения снимка

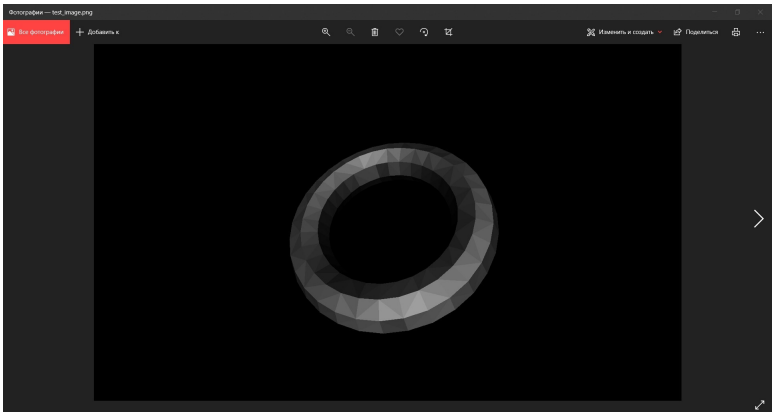

Рис. 4.6: Сохранённый снимок

Произведём тестирование функции нахождения модели из списка, наиболее подходящей к модели, изображённой на снимке. Ниже на рисунках **??—??** представлены интерфейс загрузки снимка и модель, которая была подобрана в результате работы алгоритма. В качестве снимка использовалось изображение , полученное при тестировании функции сохранения снимка.

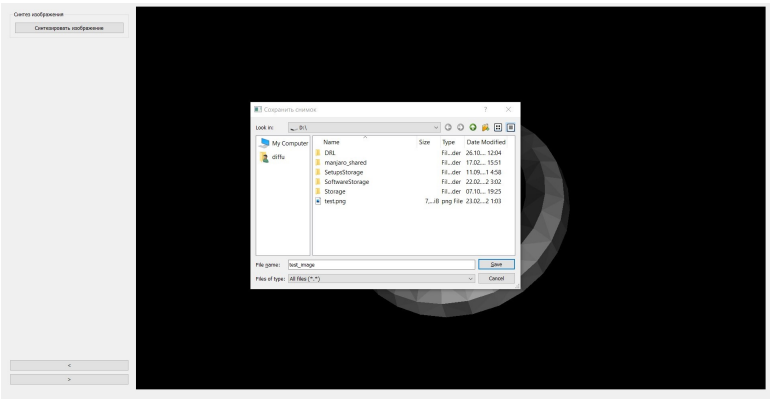

Рис. 4.7: Интерфейс загрузки снимка

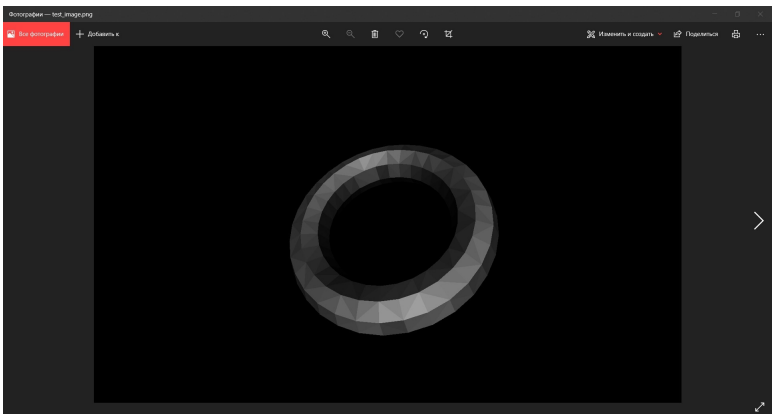

Рис. 4.8: Модель, подобранная в результате работы алгоритма

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

Целью эксперимента является исследование зависимости скорости работы алгоритма отрисовки от количества полигонов в просматриваемой модели.

Для получения результатов исследовалось изображение сферы, у которой количество полигонов принимало значения от 20 до 1280.

Результаты эксперимента представлены ниже в таблице 4.3 и на графике 4.9.

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 |

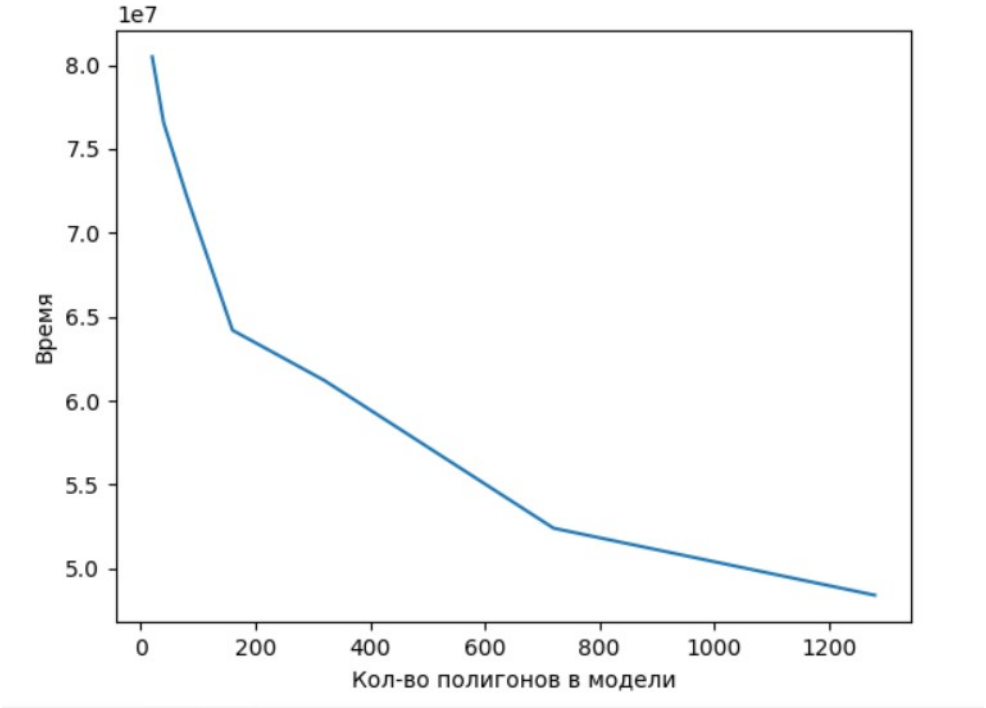


Рис. 4.9: Визуализация результатов эксперимента

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

## 4.3 Исследование зависимости скорости работы программы от размера области, которую занимает полигон на экране

Целью эксперимента являлось исследование зависимости работы алгоритма отрисовки от размера полигона.

Для получения результатов исследовалась скорость отрисовки полигона, который мастабировался так, что площадь покрытия экрана его проекцией изменялась от 0.5% до 32%.

Результаты эксперимента представлены ниже в таблице 4.2 и на графике 4.10.

Таблица 4.2: Время отрисовки в зависимости от размера полигона

| **размер полигона** | **время отрисовки (нс)** |
|---|---|
| 0.5% | 4993100.0 |
| 2.0% | 6198600.0 |
| 4.5% | 8392840.0 |
| 8.0% | 11199080.0 |
| 12.5% | 15184980.0 |
| 18.0% | 19407080.0 |
| 24.5% | 24007740.0 |
| 32.0% | 30205220.0 |

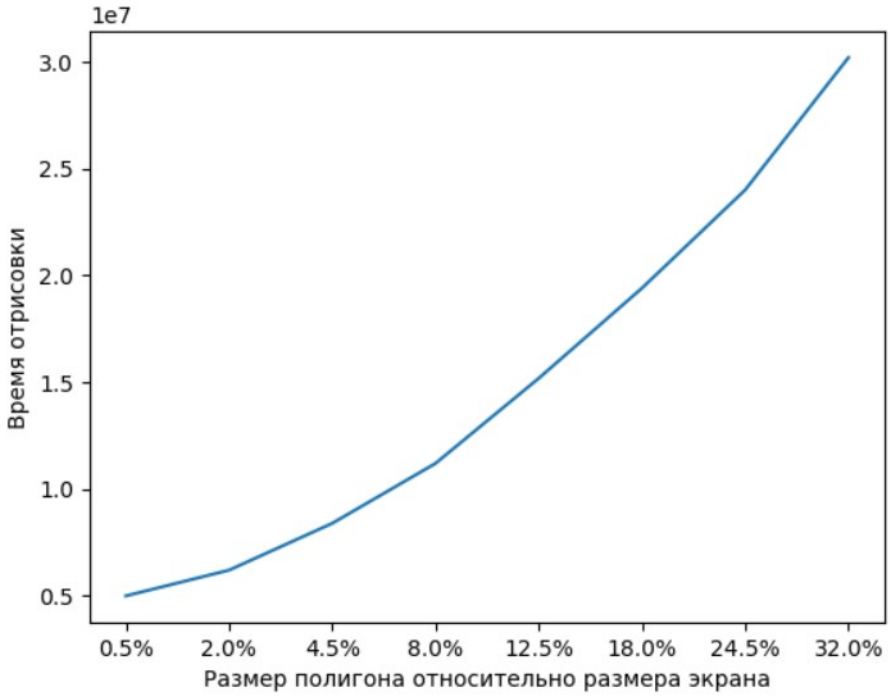


Рис. 4.10: Визуализация результатов эксперимента

## 4.4 Исследование 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.

В данном эксперименте создавался набор из 5 снимков для каждого из объектов из списка объектов, после чего данные снимки подаются на вход алгоритму. Полученный объект сравнивается с тем, который служил основой для снимка.

Результаты эксперимента представлены ниже в таблице

Таблица 4.3: 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, циллиндр, изосферу и сферу. Однако если объекты имеют похожую геометрию, то алгоритм не во всех случаях может определить корректную модель, что происходит в случае куба и пирамиды.

## 4.5 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 так, что чем больше полигонов, тем быстрее работает алгоритм отрисовки, достигая разницы более чем в 1.6 раз.

В результате эксперимента по оценке зависимости скорости работы алгоритма отрисовки от размера полигонов было выявлено, что скорость работы алгоритма отрисовки зависит от количества полигонов так, что чем больше размер проекции полигона на экран, тем медленнее работает алгоритм, достигая разницы в более чем в 7.5 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 заметно отличающуюся внешнюю форму, такие как тор, конус, циллиндр, изосфера и сфера. Однако если объекты имеют похожую геометрию, то алгоритм не во всех случаях может определить корректную модель, что происходит в случае куба и пирамиды.

# Заключение

В рамках курсового проекта реализована программа, повзоляющая загружать трёхмерную модель, редактировать масштаб, вращение и положение трёхмерной модели, выводить на экран и сохранять изображение трёхмерной модели, и определять соответствие объекта, изображённого на двумерном снимке, с одним из объектов из списка тел.

В ходе выполнения проекта были выполнены следующие задачи:

- исследованы подходы к синтезу изображений;
- исследованы способы создания двумерного снимка;
- исследованы алгоритмы нахождения трёхмерного объекта по двумерному снимку;
- описаны используемые при разработке ПО алгоритмы;
- определены средства программной реализации;
- реализованы алгоритмы отрисовки сцены;
- реализован алгоритм позволяющий определять наиболее вероятное соответствие объекта, изображённого на двумерном снимке, с одним из объектов из списка тел;
- исследована зависимость скорости алгоритма отрисовки от количества полигонов в объекте;
- исследована зависимость скорости алгоритма отрисовки от размера проекции полигона;
- исследовна эффективность работы алгоритма алгоритма определения наиболее вероятного соответствия объекта на изображении объекту из списка тел.

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 так, что чем больше полигонов, тем быстрее работает

алгоритм отрисовки, достигая разницы более чем в 1.6 раз.

В результате эксперимента по оценке зависимости скорости работы алгоритма отрисовки от размера полигонов было выявлено, что скорость работы алгоритма отрисовки зависит от количества полигонов так, что чем больше размер проекции полигона на экран, тем медленнее работает алгоритм, достигая разницы в более чем в 7.5 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 заметно отличающуюся внешнюю форму, такие как тор, конус, циллиндр, изосфера и сфера. Однако если объекты имеют похожую геометрию, то алгоритм не во всех случаях может определить корректную модель, что происходит в случае куба и пирамиды.

# 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 даных. — 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

Приложение А содержит диаграмму классов разрабатываемого программного обеспечения.

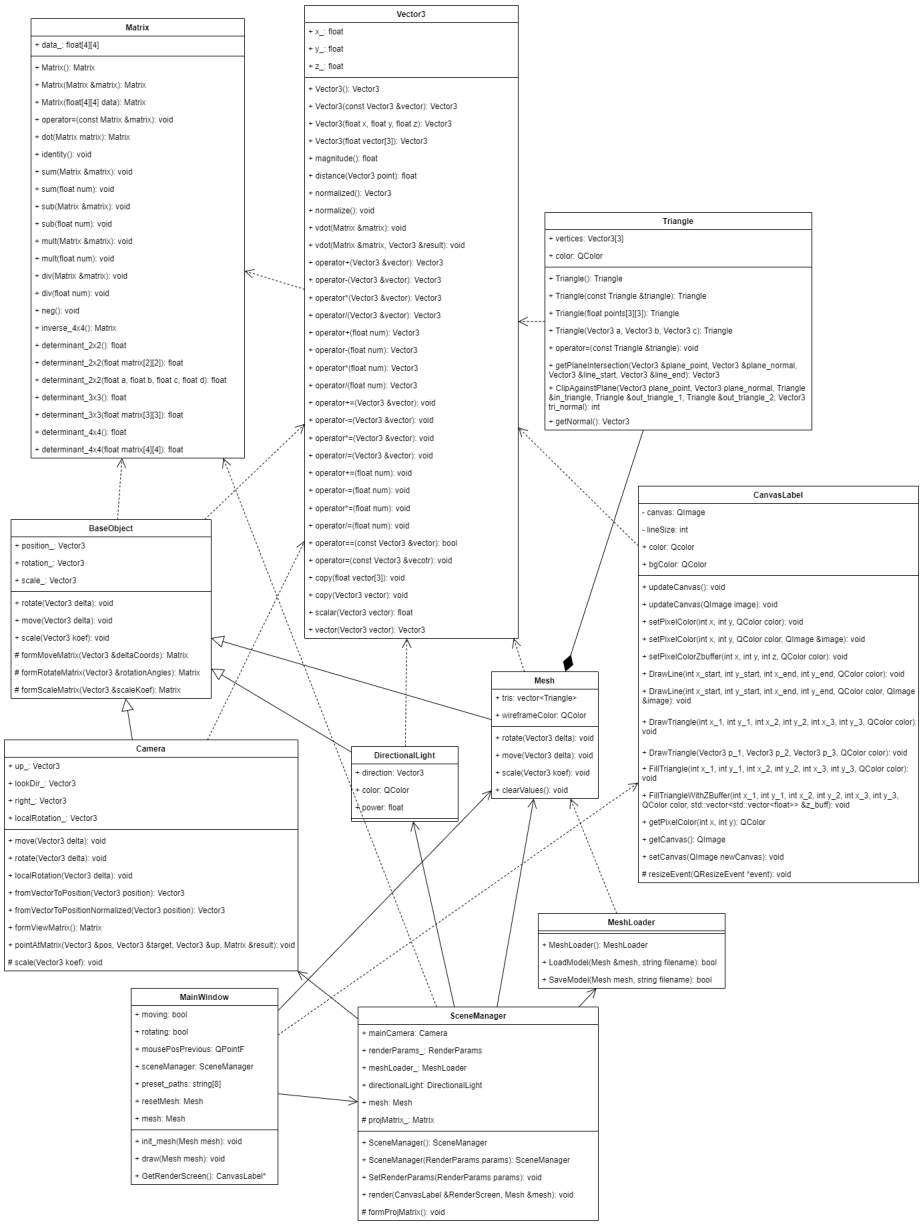


Рис. А.1: Диаграмма классов разрабатываемого программного обеспечения

# Приложение Б

Приложение Б содержит схему архитектуры разработанной нейронной сети.

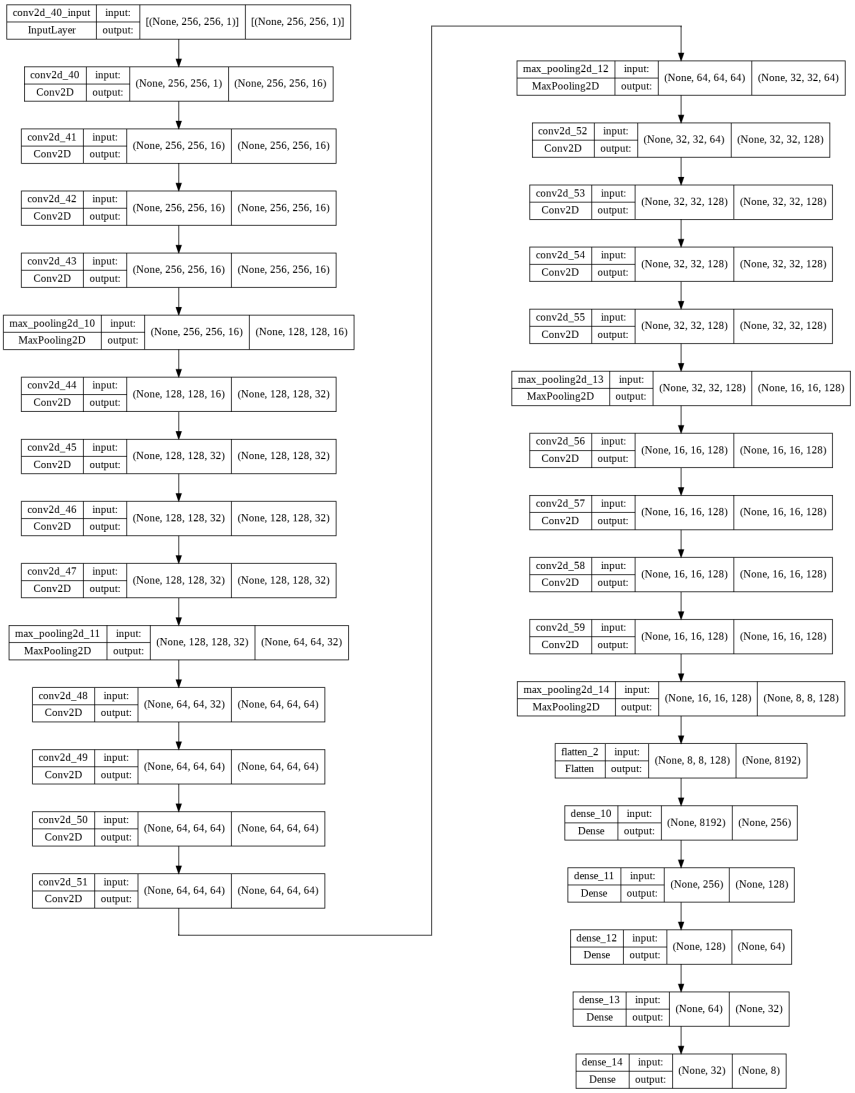


Рис. Б.1: Схема архитектуры разработанной нейронной сети

# Приложение В

Приложение В содержит схему алгоритма z-буфера.

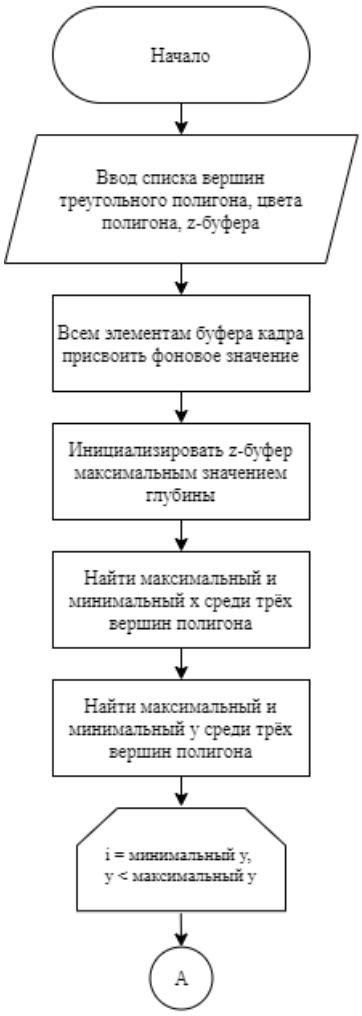


Рис. В.1: Первая часть схемы алгоритма z-буфера

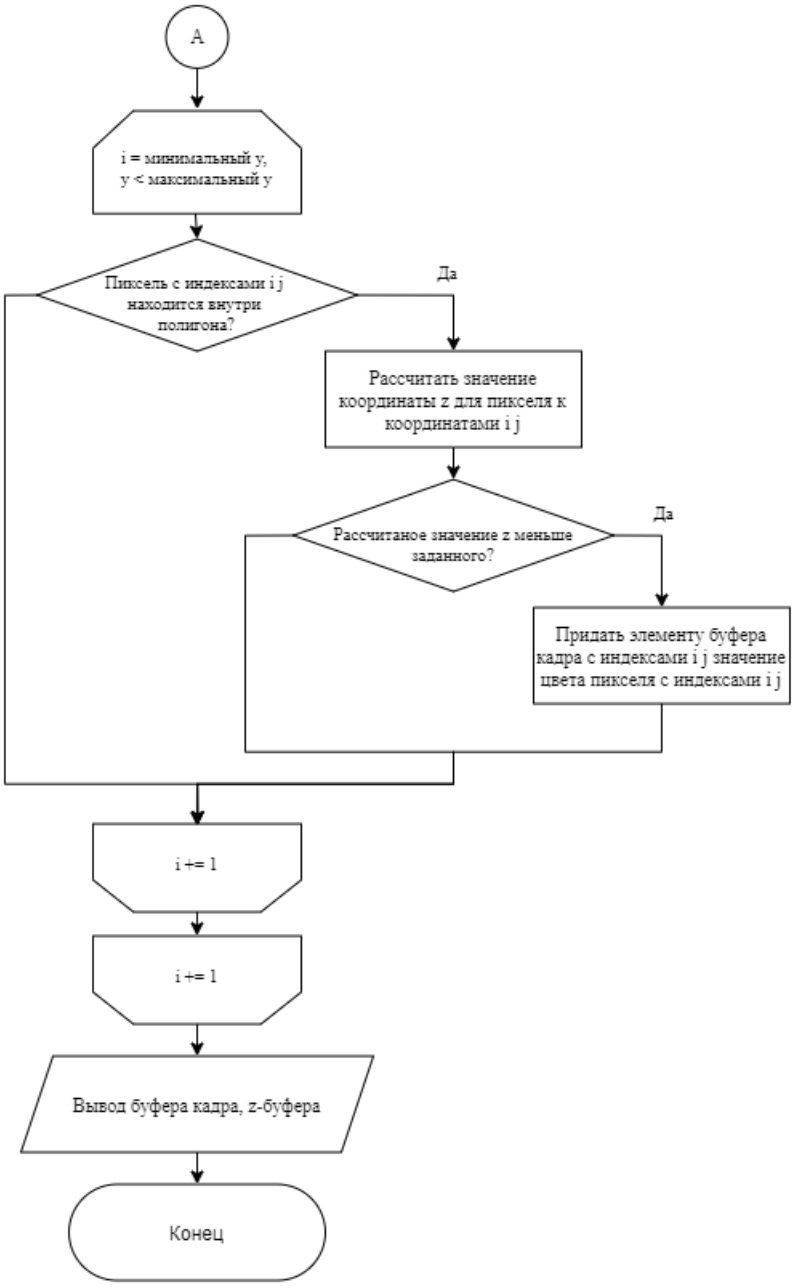


Рис. В.2: Вторая часть схемы алгоритма z-буфера

# Приложение Г

Приложение В содержит листинги реализаций основных алгоритмов.

Ниже в листинге Г.1 представлена реализация общего алгоритма отрисовки.

Листинг Г.1: Реализация общего алгоритма отрисовки

```
void SceneManager::render(CanvasLabel &RenderScreen, Mesh &mesh)
{
    Matrix viewMatrix;
    viewMatrix = mainCamera.formViewMatrix();

    std::vector<std::vector<float>> z_buff(RenderScreen.size().height(),
        std::vector<float>(RenderScreen.size().width()));

    for (auto &row : z_buff)
    {
        for (auto &elem : row)
        {
            elem = std::numeric_limits<float>::max();
        }
    }

    vector<Triangle> vecTrianglesToRaster;

    for (auto &tri : mesh.tris)
    {
        vecTrianglesToRaster.clear();

        float dp =
            directionalLight.direction.normalized().scalar(tri.getNormal());

        QColor color;
        if (dp <= 0)
        {
            color.setRgb(0, 0, 0);
        }
        else
        {
```

```
            int colorVal = dp * 255 * directionalLight.power;
            if (colorVal > 255)
                colorVal = 255;
            color.setRgb(colorVal, colorVal, colorVal);
        }
        tri.color = color;

        Triangle triangleProjected, triangleViewed;

        tri.vertices[0].vdot(viewMatrix, triangleViewed.vertices[0]);
        tri.vertices[1].vdot(viewMatrix, triangleViewed.vertices[1]);
        tri.vertices[2].vdot(viewMatrix, triangleViewed.vertices[2]);

        int clipped_num;
        Triangle clipped[2];
        clipped_num = triangleViewed.ClipAgainstPlane({0.0f, 0.0f,
            renderParams_.fNear}, {0.0f, 0.0f, 1.0f}, triangleViewed,
            clipped[0], clipped[1], triangleViewed.getNormal());

        for (int n = 0; n < clipped_num; n++)
        {
            clipped[n].vertices[0].vdot(projMatrix_,
                triangleProjected.vertices[0]);
            clipped[n].vertices[1].vdot(projMatrix_,
                triangleProjected.vertices[1]);
            clipped[n].vertices[2].vdot(projMatrix_,
                triangleProjected.vertices[2]);

            triangleProjected.vertices[0].z_ = clipped[n].vertices[0].z_;
            triangleProjected.vertices[1].z_ = clipped[n].vertices[1].z_;
            triangleProjected.vertices[2].z_ = clipped[n].vertices[2].z_;

            vecTrianglesToRaster.push_back(triangleProjected);
        }

        for (auto &triToRaster : vecTrianglesToRaster)
        {
            triToRaster.vertices[0].x_ *= 0.1 * RenderScreen.size().width();
            triToRaster.vertices[0].y_ *= 0.1 * RenderScreen.size().height();

            triToRaster.vertices[1].x_ *= 0.1 * RenderScreen.size().width();
            triToRaster.vertices[1].y_ *= 0.1 * RenderScreen.size().height();

            triToRaster.vertices[2].x_ *= 0.1 * RenderScreen.size().width();
            triToRaster.vertices[2].y_ *= 0.1 * RenderScreen.size().height();

            Triangle clipped[2];
            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,
                                (float)RenderScreen.height()/2 - 1, 0.0f }, { 0.0f,
                                -1.0f, 0.0f }, test, clipped[0], clipped[1],
                                test.getNormal());
92                          break;
93                      case 1:
94                          nTrisToAdd = test_temp.ClipAgainstPlane({ 0.0f,
                                -(float)RenderScreen.height()/2 + 1, 0.0f }, { 0.0f,
                                1.0f, 0.0f }, test, clipped[0], clipped[1],
                                test.getNormal());
95                          break;
96                      case 2:
97                          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());
98                          break;
99                      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 }
```

Ниже в листинге Г.2 представлена реализация алгоритма отсекания невидимой части треугольного полигона.

Листинг Г.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_ +
               plane_normal.z_ * p.z_ - plane_normal.scalar(plane_point));
8      };
9
10     Vector3* inside_points[3];
11     int nInsidePointCount = 0;
12     Vector3* outside_points[3];
13     int nOutsidePointCount = 0;
14
15     float d0 = dist(in_triangle.vertices[0]);
16     float d1 = dist(in_triangle.vertices[1]);
17     float d2 = dist(in_triangle.vertices[2]);
18
19     if (d0 >= 0)
20         inside_points[nInsidePointCount++] = &in_triangle.vertices[0];
21     else
22         outside_points[nOutsidePointCount++] = &in_triangle.vertices[0];
23
```

```
24      if (d1 >= 0)
25          inside_points[nInsidePointCount++] = &in_triangle.vertices[1];
26      else
27          outside_points[nOutsidePointCount++] = &in_triangle.vertices[1];
28
29      if (d2 >= 0)
30          inside_points[nInsidePointCount++] = &in_triangle.vertices[2];
31      else
32          outside_points[nOutsidePointCount++] = &in_triangle.vertices[2];
33
34      if (nInsidePointCount == 0)
35          return 0;
36
37      if (nInsidePointCount == 3)
38      {
39          out_triangle_1 = in_triangle;
40
41          if (out_triangle_1.getNormal().normalized().scalar(tri_normal) < 0)
42          {
43              Vector3 temp = out_triangle_1.vertices[2];
44              out_triangle_1.vertices[2] = out_triangle_1.vertices[1];
45              out_triangle_1.vertices[1] = temp;
46          }
47          return 1;
48      }
49
50      if (nInsidePointCount == 1 && nOutsidePointCount == 2)
51      {
52          out_triangle_1 = in_triangle;
53          out_triangle_1.vertices[0] = *inside_points[0];
54          out_triangle_1.vertices[1] = getPlaneIntersection(plane_point,
                plane_normal, *inside_points[0], *outside_points[0]);
55          out_triangle_1.vertices[2] = getPlaneIntersection(plane_point,
                plane_normal, *inside_points[0], *outside_points[1]);
56
57          if (out_triangle_1.getNormal().normalized().scalar(tri_normal) < 0)
58          {
59              Vector3 temp = out_triangle_1.vertices[2];
60              out_triangle_1.vertices[2] = out_triangle_1.vertices[1];
61              out_triangle_1.vertices[1] = temp;
62          }
63
64          return 1;
65      }
66
67      if (nInsidePointCount == 2 && nOutsidePointCount == 1)
68      {
69          out_triangle_1 = in_triangle;
```

```
70          out_triangle_2 = in_triangle;
71
72          out_triangle_1.vertices[0] = *inside_points[0];
73          out_triangle_1.vertices[1] = *inside_points[1];
74          out_triangle_1.vertices[2] = getPlaneIntersection(plane_point,
                plane_normal, *inside_points[0], *outside_points[0]);
75
76          if (out_triangle_1.getNormal().normalized().scalar(tri_normal) < 0)
77          {
78              Vector3 temp = out_triangle_1.vertices[2];
79              out_triangle_1.vertices[2] = out_triangle_1.vertices[1];
80              out_triangle_1.vertices[1] = temp;
81          }
82
83          out_triangle_2.vertices[0] = *inside_points[1];
84          out_triangle_2.vertices[1] = out_triangle_1.vertices[2];
85          if (abs(out_triangle_2.vertices[1].x_ - out_triangle_2.vertices[0].x_)
                < 0.0001 &&\
86              abs(out_triangle_2.vertices[1].y_ - out_triangle_2.vertices[0].y_)
                    < 0.0001 &&\
87              abs(out_triangle_2.vertices[1].z_ - out_triangle_2.vertices[0].z_)
                    < 0.0001)
88              out_triangle_2.vertices[1] = out_triangle_1.vertices[1];
89          out_triangle_2.vertices[2] = getPlaneIntersection(plane_point,
                plane_normal, *inside_points[1], *outside_points[0]);
90
91          if (out_triangle_2.getNormal().normalized().scalar(tri_normal) < 0)
92          {
93              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 }
```

Ниже в листинге Г.3 представлена реализация алгоритма z-буфера.

Листинг Г.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_);
```

```
    int y_min = get_min(p_1.y_, p_2.y_, p_3.y_);

    Vector3 temp_p_1;
    Vector3 temp_p_2;
    Vector3 temp_p_3;

    temp_p_1.y_ = y_max;
    if (int(temp_p_1.y_) == int(p_1.y_))
        temp_p_1 = p_1;
    if (int(temp_p_1.y_) == int(p_2.y_))
        temp_p_1 = p_2;
    if (int(temp_p_1.y_) == int(p_3.y_))
        temp_p_1 = p_3;

    temp_p_3.y_ = y_min;
    if (int(temp_p_3.y_) == int(p_1.y_) && int(temp_p_1.y_) != int(p_1.y_))
        temp_p_3 = p_1;
    if (int(temp_p_3.y_) == int(p_2.y_) && int(temp_p_1.y_) != int(p_2.y_))
        temp_p_3 = p_2;
    if (int(temp_p_3.y_) == int(p_3.y_) && int(temp_p_1.y_) != int(p_3.y_))
        temp_p_3 = p_3;

    if ((temp_p_1 == p_1 && temp_p_3 == p_2) || (temp_p_1 == p_2 && temp_p_3 ==
       p_1))
        temp_p_2 = p_3;
    if ((temp_p_1 == p_1 && temp_p_3 == p_3) || (temp_p_1 == p_3 && temp_p_3 ==
       p_1))
        temp_p_2 = p_2;
    if ((temp_p_1 == p_2 && temp_p_3 == p_3) || (temp_p_1 == p_3 && temp_p_3 ==
       p_2))
        temp_p_2 = p_1;

    for (int y = y_min; y < y_max; y++)
    {
        for (int x = x_min; x < x_max; x++)
        {
            if (abs(x) < this->size().width()/2 && abs(y) <
               this->size().height()/2)
            {
                float e21 = TestPoint(x, y, p_2.x_, p_2.y_, p_1.x_, p_1.y_);
                float e32 = TestPoint(x, y, p_3.x_, p_3.y_, p_2.x_, p_2.y_);
                float e13 = TestPoint(x, y, p_1.x_, p_1.y_, p_3.x_, p_3.y_);

                if (e21 >= 0.0f && e32 >= 0.0f && e13 >= 0.0f)
                {
                    if (y == y_max || y == y_min)
                    {
                        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 +
                                    this->size().width()/2])
74                              {
75                                  setPixelColor(x, y, color);
76                                  z_buff[this->size().height()/2 - y][x +
                                        this->size().width()/2] = z;
77                              }
78                              //setPixelColor(x, y, color);
79                          }
80                          else if (y == int(temp_p_3.y_))
81                          {
82                              float z = temp_p_3.z_;
83                              if (z < z_buff[this->size().height()/2 - y][x +
                                    this->size().width()/2])
84                              {
85                                  setPixelColor(x, y, color);
86                                  z_buff[this->size().height()/2 - y][x +
```

```
                        this->size().width()/2] = z;
87                  }
88                  //setPixelColor(x, y, color);
89              }
90          }
91          else if (y >= int(temp_p_2.y_))
92          {
93              int x_a = temp_p_1.x_ + (temp_p_2.x_ - temp_p_1.x_)*((y
                    - temp_p_1.y_)/(temp_p_2.y_ - temp_p_1.y_));
94              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_));
95              float z_a = temp_p_1.z_ + (temp_p_2.z_ -
                    temp_p_1.z_)*((y - temp_p_1.y_)/(temp_p_2.y_ -
                    temp_p_1.y_));
96              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_));
97
98              if (x_a != x_b)
99              {
100                 float z = z_a + (z_b - z_a)*((x - x_a)/(x_b - x_a));
101
102                 if (z < z_buff[this->size().height()/2 - y][x +
                        this->size().width()/2])
103                 {
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_));
                        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_));

                        if (x_a != x_b)
                        {
                            float z = z_a + (z_b - z_a)*((x - x_a)/(x_b - x_a));

                            if (z < z_buff[this->size().height()/2 - y][x +
                                this->size().width()/2])
                            {
                                setPixelColor(x, y, color);
                                z_buff[this->size().height()/2 - y][x +
                                    this->size().width()/2] = z;
                            }
                            //setPixelColor(x, y, color);
                        }
                        else
                        {
                            if (z_a < z_buff[this->size().height()/2 - y][x +
                                this->size().width()/2])
                            {
                                setPixelColor(x, y, color);
                                z_buff[this->size().height()/2 - y][x +
                                    this->size().width()/2] = z_a;
                            }
                        }
                    }
                }
            }
        }
    }
}
```

Ниже в листинге Г.4 представлена реализация алгоритма обучения свёрточной нейросети.

Листинг Г.4: Реализация алгоритма обучения свёрточной нейросети

```
import tensorflow as tf
from tensorflow import keras
from keras import layers
import numpy as np
import cv2

train_materials_path =
    "D:/Storage/Andrew/BMSTU/BMSTU/CW/Work/CourseProject/TrainMaterials/"
```

```
8
9  import os, os.path
10
11 x_train = []
12 y_train = []
13
14 #size of dataset
15 dataset_size = len(os.listdir(train_materials_path))
16 classes = 8
17 image_size = (256, 256)
18 image_shape = (256, 256, 1)
19 batch_size = 128
20
21 import os
22
23 clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
24 count = 0
25
26 for name in os.listdir(train_materials_path):
27   fpath_img = train_materials_path + name
28   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
```

```
from keras import metrics

epochs = 20

opt = tf.keras.optimizers.Adam(learning_rate=0.001, decay=1e-3 / 200)

callbacks = [
    keras.callbacks.ModelCheckpoint("save_model_at_{epoch}.h5"),
]
model.compile(loss=losses.categorical_crossentropy, optimizer=opt,
    metrics=metrics.Accuracy())

history = model.fit(
    x_train, y_train, epochs=epochs, callbacks=callbacks, validation_split=0.2,
)

model.save('new_model.h5')
```

Ниже в листинге Г.5 представлена реализация общего алгоритма отрисовки.

Листинг Г.5: Реализация алгоритма определения трёхмерного объекта наиболее вероятно являющимся объектом изображённым на снимке

```
def predict_model(fpath_img: str) -> int:
    import cv2
    import numpy as np
    import keras
    import tensorflow as tf
    from keras import layers

    try:
        image_size = (256, 256)

        clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
        image = cv2.imread(fpath_img, cv2.IMREAD_GRAYSCALE)

        max_index = 0
        count = 0
        max_val = 0

        min_dim = min(image.shape)
        w = min_dim
        h = min_dim
        center = image.shape
        x = center[1]/2 - w/2
        y = center[0]/2 - h/2

        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
```

# Приложение Д

Приложение Д содержит снимки используемых при обучении нейросети моделей.

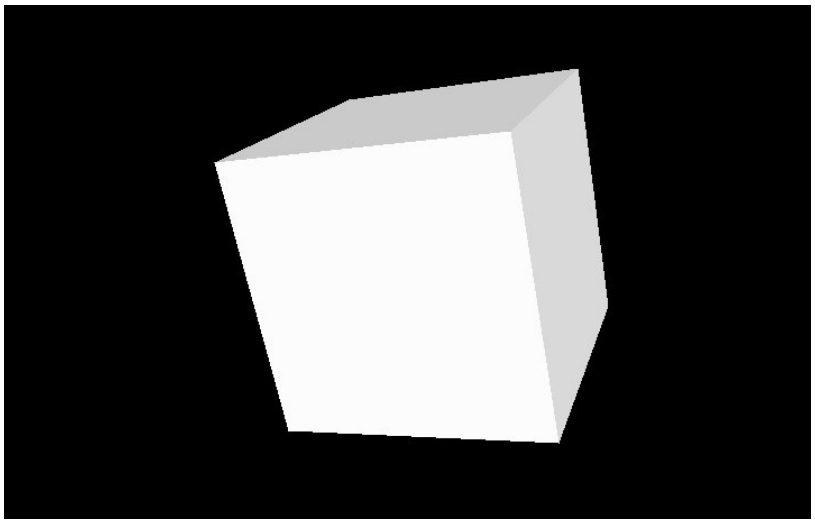

Рис. Д.1: Куб

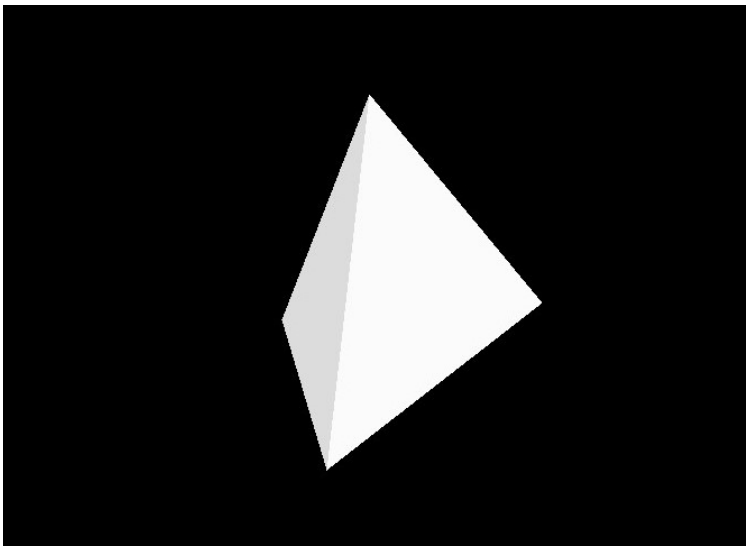

Рис. Д.2: Тетраэдр

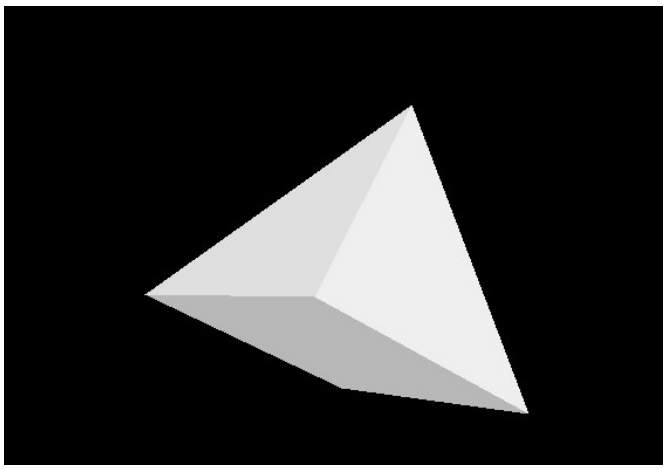

Рис. Д.3: Пирамида

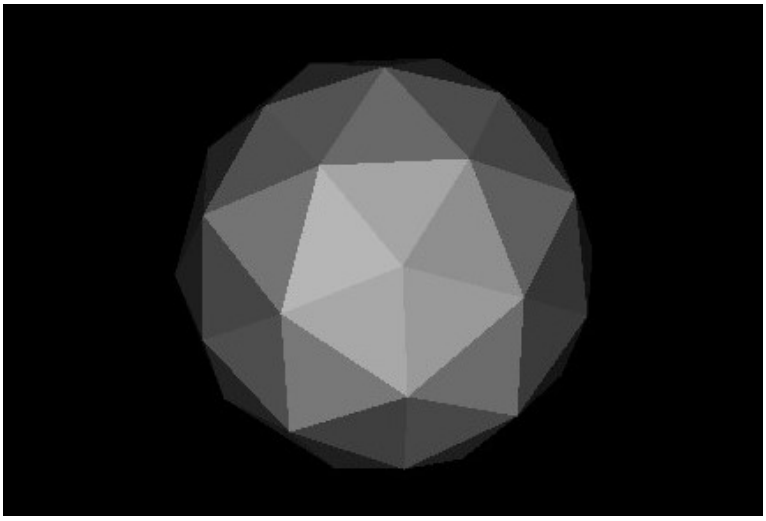

Рис. Д.4: Изосфера

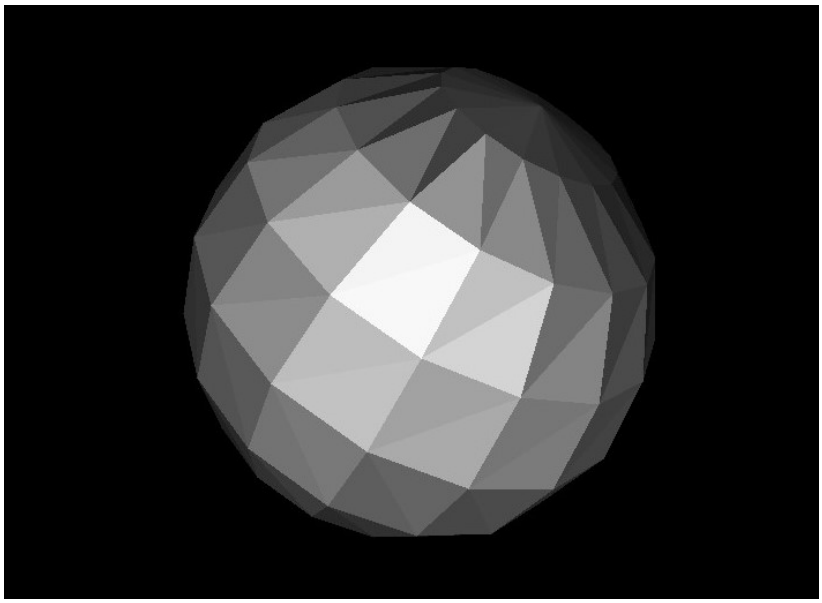

Рис. Д.5: Сфера

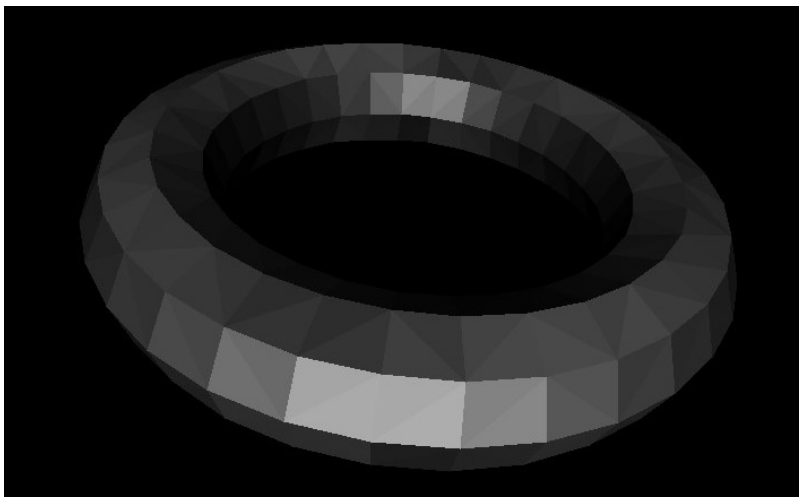

Рис. Д.6: Тор

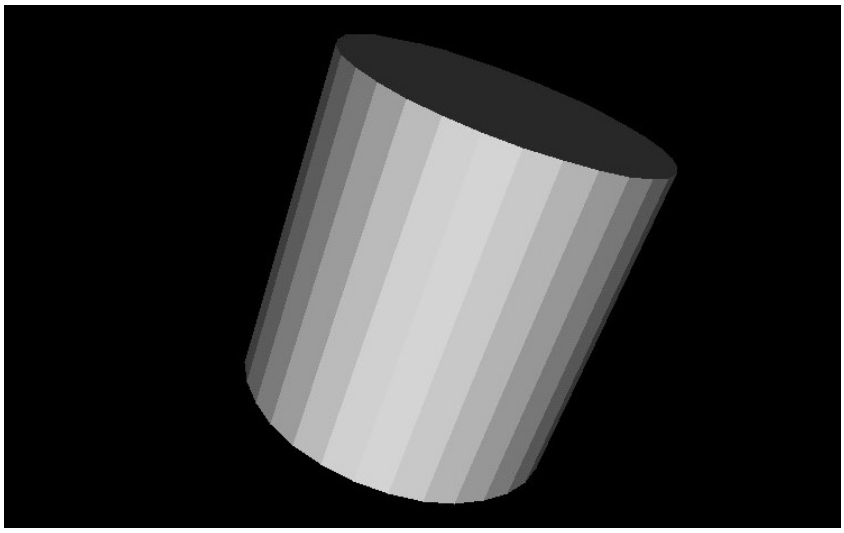

Рис. Д.7: Циллиндр

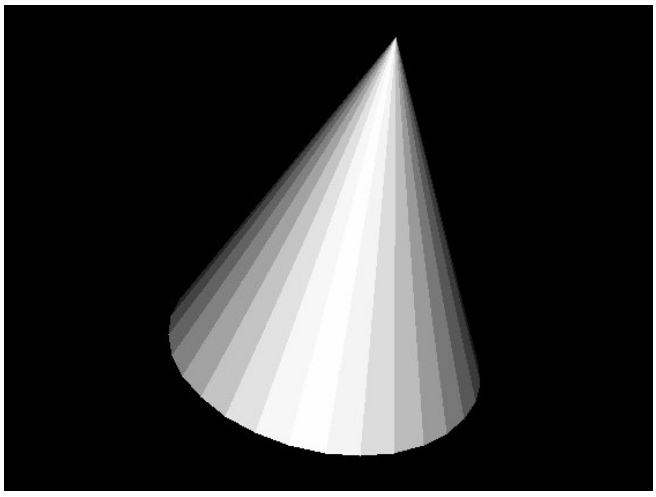

Рис. Д.8: Конус