|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **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»

**ОТЧЕТ ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ**

Студент Бугаенко Андрей Павлович

*фамилия, имя, отчество*

Группа ИУ7-45Б

Тип практики технологическая

Название предприятия МГТУ им. Н. Э. Баумана, каф. ИУ7

Студент **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** Бугаенко А.П.

*подпись, дата фамилия, и.о.*

Руководитель практики **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

*подпись, дата фамилия, и.о.*

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Москва, 2021 г.*

**Оглавление**

Введение 3

1. Аналитический раздел 5

1.1 Формализация объектов сцены 3D редактора 5

1.2 Анализ способа представления поверхности 6

1.3 Анализ алгоритмов удаления невидимых линий и поверхностей 8

1.3 Анализ алгоритмов удаления невидимых линий и поверхностей 8

1.4 Анализ методов закрашивания 10

1.5 Анализ алгоритмов построения теней 11

1.6 Анализ алгоритмов ML для анализа изображения 11

2. Конструкторский раздел 14

2.1 Общий алгоритм решения задачи 14

2.2 Алгоритм закраски 15

2.3 Алгоритм z-буфера 15

2.4 Простой метод освещения 16

2.5 Алгоритм трассировки лучей 16

2.6 Алгоритм формирования трёхмерной модели на основе её изображения 17

2.7 Выбор используемых типов и структур данных 18

3. Технологический раздел 20

3.1 Выбор и обоснование языка программирования и среды разработки 20

3.2 Структура и состав классов 22

Заключение 29

Список литературы 30

**Введение**

В наше время компьютерная графика широко применяется во многих сферах жизни человеческого общества для визуального представления различных видов информации. В частности 3D графика часто используется для создания изображений или видео путём моделирования объектов в трёхмерном пространстве. С помощью 3D графики люди получили возможность достаточно точно симулировать реальные объекты и различные взаимодействия между ними. Однако на сегодняшний день всё сводится к тому, что человек получает визуальную информацию в виде двухмерного изображения. Поэтому задача трёхмерного моделирования сводится к получению двухмерных проекций трёхмерного пространства, которые формируются при помощи сложных взаимодействий геометрии моделей с различными источниками света. Процесс генерации двумерного изображения из трёхмерной модели называется рендернигом. На данный момент рендеринг изображений является хорошо изученным процессом, развитие которого продолжается уже несколько десятков лет. На данный момент существует обширная выборка различных методов рендеринга трёхмерных изображений, делящаяся на 4 основные группы: методы, основывающиеся на растеризации, методы на основе технологии рейкастинга, методы на основе технологии трассировки лучей, методы на основе трассировки путей. Однако вся задача всё равно сводится к тому, чтобы каким-то образом создать реалистичную проекцию некого трёхмерного пространства, состоящего из трёхмерных моделей. И на данный момент создание трёхмерных моделей является одной из самых трудоёмких частей трёхмерного моделирования, поскольку для данного процесса требуется специально обученные специалисты и довольно много времени и сил. В общем случае задача этого специалиста сводится к тому, чтобы взять некий объект (не обязательно реальный), и на его основе построить его трёхмерную модель. Автоматизация данного подхода позволила бы сэкономить и деньги и время. В качестве одного из таких способов можно рассматривать применение технологий машинного обучения — с помощью свёрточной нейросети мы обрабатываем изображение и на выходе получаем набор полигонов, составляющих распознанную модель. Поэтому целью данной курсовой работы является создание программы, способной обработать изображение, и на выходе предоставить трёхмерную модель во встроенном редакторе с возможностью выгрузки этой модели в файл для дальнейшей работы. Для того, чтобы реализовать представленное выше решение необходимо решить две задачи. Первая — создание редактора 3D моделей, имеющего набор базовых команд (перемещение, вращение, масштаб) и возможность создавать изображения находящейся на сцене модели. Вторая — выбор подходящего алгоритма машинного обучения, реализация его в виде встроенной функции редактора и обучение на основе данных, сгенерированных с помощью реализованного ранее редактора 3D моделей.

**1. Аналитический раздел**

**1.1 Формализация объектов сцены 3D редактора**

3D редактор позволяет пользователю взаимодействовать со сценой, которая может содержать в себе следующие объекты:

* Направленный источник света — задаётся единичным вектором, начало которого совпадает с началом координат, и действительным числом, задающим интенсивность свечения. Цвет распространяемого света задаётся через RGB параметры.
* Трёхмерный объект — трёхмерная модель загружаемая либо из встроенной в программу базы, либо из внешнего объектного файла.
* Камера — камера в пространстве сцены, которая задаётся с помощью таких параметров, как:
  + Соотношение сторон экрана — одно действительное положительное число.
  + Ближняя граница прорисовки — одно действительное положительное число.
  + Дальняя граница прорисовки — одно действительное положительное число
  + Поле видимости — задаётся в градусах, одно действительное положительное число.

Также к параметрам камеры относятся вектора, отвечающие за её ориентацию в пространстве, это:

* + Вектор направления — указывает, в какую сторону на данный момент направлена камера. Задаётся как вектор с компонентами x, y, z в трёхмерном декартовом пространстве.
  + Правый вектор — указывает на направление «право» относительно камеры.
  + Верхний вектор — указывает на направление «вверх» относительно камеры.

**1.2 Анализ способа представления поверхности**

При рассмотрении того, каким образом можно задать некоторую поверхность в пространстве редактора выделяется три основных способа:

* Каркасный — модель представляется в виде множества вершин, связанных между собой рёбрами.
* Поверхностный — модель представляется в виде множества поверхностей.
* Объёмный — модель формируется из элементарных объектов (базисных тел) с использованием логических операций объединения, вычитания, пересечения.

В нашем случае наилучший способ описания модели — поверхностный, так как каркасные модели попросту не могут обладать достаточным количеством деталей (в силу отсутствия поверхности свет не может быть преломлён и как следствие мы не сможем извлечь из двухмерной проекции каркасной модели никакой информации о её трёхмерной структуре) необходимых для восстановления модели по её снимку. Объёмная модель также не имеет смысла, поскольку мы работаем не с объёмом модели, а с её поверхностью и затраты по памяти в этом случае только увеличатся.

Поверхностную модель мы можем задать следующими способами:

Параметрическое представления — поверхность определяется некой функцией, зависящей от одного или нескольких параметров. В основном данный метод используется для задания в трёхмерном пространстве объектов вращения. Для нашего случая этот метод не подходит, так как представление в сцене объектов, не являющихся объектами вращения будет затруднительно.

Полигональная сетка — в этом случае поверхность задаётся совокупностью вершин, рёбер и граней, которые определяют внешний вид объекта. Мы будем использовать этот способ в силу его универсальности и повсеместного применения в современном 3D моделировании (самые популярные редакторы трёхмерных объектов используют полигональные сетки в качестве представления поверхностной модели). В свою очередь полигональная сетка может быть реализована несколькими разными способами:

* Вершинное представление — в этом случае формируется список, в котором для каждой вершине ставятся в соответствие вершины, с которыми она соединена рёбрами. Однако этот способ довольно громоздкий, так как для того, чтобы сформировать грани приходится проходить под данным несколько раз.
* Таблица углов — в данном случае вершины хранятся в предопределённой таблице. В этой задаче использоваться этот способ не будет, так как изменение данных затратно по времени.
* Список граней — представляет объект как множество граней и вершин.
* Список полигонов — представляет объект как множество поверхностей, формирующих поверхность модели.

В нашем случае мы будем использовать список треугольных полигонов, которые в совокупности будут составлять поверхность модели. Данный способ будет наиболее подходящим в нашем случае, так как треугольник в пространстве однозначно задаёт плоскость, вследствие чего алгоритмы работы с ним намного легче, чем с произвольными полигонами (которые в большей части случаев всё равно приходится разбивать на треугольники).

**1.3 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 формирование данных для обучения модели машинного обучения на основе снимков редактора. Вследствие этого нам необходим алгоритм, способный произвести удаление поверхностей достаточно точно для того, чтобы модель смогла распознать объект. При этом не обязательно использование сложных алгоритмов, которые при больших затратах вычислительных мощностей дают примерно тот же результат.

Алгоритм обратной трассировки лучей — данный метод изначально рассчитан на построение реалистичного изображения со сложной моделью освещения. Принцип обратной трассировки лучей состоит в том, что через каждую точку экрана строится обратный луч света до пересечения с ближайшим объектом сцены, далее из этой точки проводится луч в направлении источника света, таким образом, моделируется распространение света. Если луч к источнику света, ничего не пересекает на своём пути, значит данная точка освещена, иначе она лежит в тени. Если луч попадает на зеркальную поверхность, то, в соответствии с законами оптики, выпускается отражённый луч, что даёт возможность построить отражение. В зависимости от свойств среды, через которую проходит луч, он может преломляться, что позволяет моделировать сложные реалистичные световые эффекты. Этот метод позволяет получить не только тени от объектов, но и рассчитать вторичное освещение, когда отражённый тусклый свет попадает в непосредственно затенённые области и размывает тени.

Алгоритм Робертса — это метод, работающий в объектном пространстве. Алгоритм прежде всего удаляет из каждого тела те ребра или грани, которые экранируются самим телом. Затем каждое из видимых ребер каждого тела сравнивается с каждым из оставшихся тел для определения того, какая его часть или части, если таковые есть, экранируются этими телами. Поэтому вычислительная трудоемкость алгоритма Робертса растет теоретически, как квадрат числа объектов. Преимуществом данного алгоритма является точность вычислений. Она достигается за счет работы в объектном пространстве, в отличии от большинства других алгоритмов. Некоторые из оптимизаций крайне сложны, что затрудняет реализацию этого алгоритма.

Алгоритм Варнока — этот алгоритм основывается на рекурсивном разбиении экрана. В зависимости от расположения объектов это может стать, как положительной, так и отрицательной стороной алгоритма. Чем меньше пересечений объектов, тем быстрее алгоритм завершит свою работу.

Алгоритм, использующий Z-буфер — данный алгоритм работает в пространстве изображения и использует z-буфер для рисования сложных поверхностей. Z-буфер - это отдельный буфер глубины, используемый для запоминания координаты z или глубины каждого видимого пикселя в пространстве изображения. В процессе работы глубина или значение z каждого нового пикселя, который нужно занести в буфер кадра, сравнивается с глубиной того пикселя, который уже занесен в z-буфер. Если это сравнение показывает, что новый пиксель расположен впереди пикселя, находящегося в буфере кадра, то новый пиксель 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 х и у наибольшего значения функции z (х, у).

В нашей задаче важна не скорость построения изображения, а его качество, выражающееся в правильном удалении всех невидимых линий и поверхностей модели (особенно это нужно учитывать в случае, когда модель имеет сложную форму и несколько раз перекрывает сама себя), поэтому мы будем использовать алгоритм трассировки лучей для удаления невидимых частей модели в режиме рендера изображения и z-буффер в режиме просмотра модели.

**1.4 Анализ методов закрашивания**

Простая закраска —при использовании этого алгоритма вся плоскость закрашивается одним уровнем интенсивности, который вычисляется с помощью закона Ламберта (I = Iн · cos φ). Данный метод крайне прост в реализации и не требователен к ресурсам, однако не подходит для тел вращения и симуляции отражённого света.

Закраска по Гуро — алгоритм закраски поверхности по Гуро основывается на билинейной интерполяции интенсивностей. За счёт этого устраняется дискретность изменения интенсивности и создаётся иллюзия гладкой криволинейной поверхности. Данный способ закраски хорошо сочетается с диффузным отражением.

Закраска по Фонгу — алгоритм закраски поверхности по Фонгу основывается на билинейной интерполяции векторов нормалей в разных точках поверхности. За счёт этого достигается лучшая локальная аппроксимация кривизны поверхности. Получающееся изображение получается более реалистичным. Данный метод позволяет симулировать реалистичные зеркальные блики.

В нашем случае мы будем использовать простую закраску, так как при использовании интерполяции поверхности в том или ином виде теряются важные детали (становится не видна ориентация полигонов относительно источника света и других полигонов).

**1.5 Анализ алгоритмов построения теней**

Алгоритм трассировки лучей — при использовании этого алгоритма тени для объектов сцены вычисляются «автоматически», так как закрашиваются только освещённые пиксели. В нашем случае реализуется именно этот алгоритм, поэтому отдельных механизмов построения теней создавать не нужно.

В режиме просмотра модели тени, который использует алгоритм z-буффера, тени не отображаются.

**1.6 Анализ алгоритмов ML для анализа изображения**

На данный момент только начинается исследование применения алгоритмов машинного обучения для преобразования изображения в трёхмерную модель. Поэтому алгоритм придётся разрабатывать без опоры на уже существующие работы. Постановка с задачи с точки зрения машинного обучения выглядит следующим образом — на вход подаётся массив из пикселей, заданных в цветовой схеме RGB, на выходе необходимо получить набор полигонов, которые наиболее лучшим образом отображают трёхмерную модель, изображение которой было подано на вход. В качестве решения данной проблемы предлагается использовать комбинацию из нескольких свёрточных нейросетей.

Свёрточная нейросеть - специальная архитектура искусственных нейронных сетей нацеленная на эффективное распознавание образов. Данная архитектура использует некоторые особенности зрительной коры[, в которой были открыты так называемые 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 (англ. convolution layers) и субдискретизирующих слоёв (англ. subsampling layers или англ. pooling layers, слоёв подвыборки). Структура сети — однонаправленная (без обратных связей), принципиально многослойная. Для обучения используются стандартные методы, чаще всего метод обратного распространения ошибки. Функция активации нейронов (передаточная функция) — любая, по выбору исследователя. Название архитектура сети получила из-за наличия операции свёртки, суть которой в том, что каждый фрагмент изображения умножается на матрицу (ядро) 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 (последовательности карт признаков), фильтруя маловажные детали и выделяя существенное. Мы будем использовать данный тип нейронной сети, так как свёрточные нейросети предназначены для работы с изображениями, что и требуется в нашей задачи.

В качестве решения предлагается искать на изображении треугольные полигоны, в итоге создавая массив из трёх вершин, создающих трёхмерную поверхность. Алгоритм будет состоять из двух частей. В первой нам необходимо сформировать массив точек, для этого формируется сеть, на вход которой подаётся специальным образом сжатое изображение (разрешение изображения, подающегося на вход нейросети, является фиксированным, поэтому исходное изображение придётся либо обрезать и сжимать, если оно больше, чем нужно, либо наоборот расширять, если оно меньше), а также координаты точки x и y, для которых мы определяем координату z, и вероятность того, существует ли точка с этими координатами на сцене. После этого мы анализируем с помощью нейросети каждый пиксель изображения и получаем набор точек, который затем проверяется на совпадения координат и передаётся на следующий этап. Второй этап ищет грани примерно по такой же схеме. На вход подаётся то же подготовленное изображение и координаты двух точек x1, y1, z1 и x2, y2, z2. На выходе возвращается вероятность того, что между этими двумя точка существует грань. После того, как мы анализируем таким образом все комбинации точек в полученном списке точек. Мы получаем список рёбер, который затем проверяется на совпадения рёбер по координатам. После чего мы анализируем рёбра, и собираем из них треугольные полигоны, сравнивая концы отрезков — если концы разных отрезков совпадают, то это значит, что три точки, из которых состоят эти отрезки, формируют полигон. Таким образом мы проверяем все рёбра и на выходе получаем список полигонов, который можно интерпретировать как трёхмерную модель.

**2. Конструкторский раздел**

**2.1 Общий алгоритм решения задачи**

1. Задать объекты сцены
2. Задать камеру (наблюдателя)
3. Задать источники света
4. Если был выбран режим просмотра, то выполнить пункты 4.1 — 4.2
   1. Используя алгоритм z-буфера получить изображение сцены
   2. Отобразить изображение
5. Если был выбран режим создания качественного изображения, то выполнить пункты 5.1 — 5.3
   1. Используя алгоритм трассировки лучей получить изображение сцены
   2. Отобразить изображение сцены
   3. Запросить подтверждение пользователя на сохранение изображения в файловой системе компьютера
6. Если был выбран режим создания модели из изображения, то выполнить пункты 6.1 -
   1. Запросить у пользователя путь до папки, в которой находится файл, содержащий изображение
   2. Считать изображение из файла в объект, предназначенный для хранения изображения
   3. Подготовить полученные данные для анализа алгоритмами машинного обучения
   4. С помощью алгоритма получения модели сформировать новый трёхмерный объект
   5. Перейти в режим просмотра, и отобразить полученный из изображения объект на сцене

**2.2 Алгоритм закраски**

Простой алгоритм закраски треугольного полигона реализуется при помощи использования результатов векторного произведения. Для определения того, находится ли точка внутри треугольника вычисляются три векторных произведения для каждого ребра. Каждое векторное произведение рассматривает векторы, начало которых лежит в точках, образующих ребро, а конец в проверяемой точке. Пусть вектор из начальной точки будет a, а из конечной b, тогда по знаку результата выражения a\_x \* b\_y – a\_y \* b\_x мы сможем узнать, с какой стороны от вектора, образуемого из начальной и конечной вершин ребра, находится точка. Если меньше нуля, то справа, если больше, то слева. Алгоритм выглядит следующим образом:

Входные данные алгоритма — координаты точек входа x\_1, y\_1, x\_2, y\_2, x\_3, y\_3; цвет закраски — color.

1. Определяются координаты x\_max, y\_max, x\_min, y\_min ограничивающего треугольник прямоугольника.
2. В цикле от y = y\_min до y\_max с шагом 1 выполнять:
   1. В цикле от x = x\_min до x\_max с шагом 1 выполнять:
      1. Определяем положение точки относительно ребра x\_1, y\_1
      2. Определяем положение точки относительно ребра x\_2, y\_2
      3. Определяем положение точки относительно ребра x\_3, y\_3
      4. Если точка находится слева относительно всех рёбер, то закрашиваем пиксель с координатами (x, y)
      5. Иначе идём дальше по циклу

**2.3 Алгоритм z-буфера**

Входные данные алгоритма — цвет закраски изображения color.

1. Присвоить всем элементам кадра одинаковое фоновое значением
2. Инициализировать z-буфер минимальным значением глубины
3. Выполнить растровую развёртку каждого треугольного полигона сцены:
   1. Для каждого пикселя растрируемого полигона вычислить глубину z(x, y)
   2. Сравнить глубину пикселя со значением, хранящимся в z-буфере. Если z(x, y) > zбуф(x, y), то zбуф(x, y) = z(x, y), цвет пикселя (x, y) = color.
4. Отобразить результат

**2.4 Простой метод освещения**

Простой метод освещения поверхности использует для расчёта интенсивности закон Ламберта, который выглядит следующим образом:

I = I0 · cos(ɑ), где

I – результирующая интенсивность света.

I0 — интенсивность источника.

ɑ — угол между нормалью к поверхности и вектором направления света.

**2.5 Алгоритм трассировки лучей**

1. В цикле от y = 0, y < высота экрана, с шагом один выполнять:
   1. В цикле от x = 0, x < ширина экрана, с шагом один выполнять:
      1. Рассчитать параметры луча, испускаемого камерой для пикселя с координатами x, y.
      2. В цикле по всем полигонам сцены определить ближайший пересекаемый полигона
      3. Если пересечения нет, то пиксель закрашивается цветом фона и цикл переходит на следующий шаг
      4. Иначе запомнить координаты x’, y’, z’ пересечения
      5. Из точки пересечения пустить в сторону источника света луч
      6. В цикле по всем полигонам сцены проверить пересечение луча с полигоном
      7. Если есть — пиксель с координатами x, y закрашивается цветом фона
      8. Иначе пиксель закрашивается цветом с учётом интенсивности источника
2. Отобразить изображение

**2.6 Алгоритм формирования трёхмерной модели на основе её изображения**

Входные параметры алгоритма:

Порог вероятности для первой нейросети — p\_1.

Порог вероятности для второй нейросети — p\_2.

1. Запросить у пользователя изображение для обработки
2. Если изображение успешно загружено — выполнение алгоритма продолжается, иначе — алгоритм заканчивает работу
3. Приведение изображения к виду матрицы размера (w, h, 3), где w – ширина изображения, h – высота изображения, 3 — количество параметров цвета пикселя в схеме RGB
4. В цикле от y = 0 пока y < h с шагом 1 выполнять:
   1. В цикле от x = 0 пока x < w с шагом 1 выполнять:
      1. Вычисление первой моделью машинного обучения координаты z для x и y и вероятности того, что эта точка действительно существует в этих координатах.
      2. Если полученная вероятность больше чем p\_1, то точка записывается в массив вершин
      3. Иначе переход на следующий шаг
5. В цикле от y = 0 пока y < h с шагом 1 выполнять:
   1. В цикле от x = 0 пока x < w с шагом 1 выполнять:
      1. Вычисление второй моделью машинного обучения вероятности того, что между двумя точками существует ребро
      2. Если полученное значение больше чем p\_2, то ребро записывается в массив вершин.
6. В цикле для r\_1 = ребро из массива рёбер выполнять:
   1. В цикле для r\_2 = ребро из массива рёбер выполнять:
      1. Если r\_1 != r\_2 (хотя бы одна пара вершин не совпадает)
         1. Если хотя бы одна пара вершин совпадает, то:
            1. Сформировать треугольный полигон из вершин рёбер r\_1 и r\_2 и добавить его в массив треугольных полигонов
7. Если массив треугольных полигонов не пуст:
   1. Перейти в режим просмотра объектов
   2. Поместить сформированный объект на сцену
   3. Сформировать изображение сцены
8. Конец алгоритма

**2.7 Выбор используемых типов и структур данных**

Источник света — задаётся позицией в пространстве сцены, направленностью света, функцией затухания в зависимости от расстояния (линейная, квадратичная, отсутствует).

Объект сцены — задаётся как список треугольных полигонов.

Камера — задаётся позицией в пространстве сцены, направлением взгляда.

Математические абстракции:

Точка — хранит координаты x, y, z.

Вектор — хранит направление по x, y, z.

Треугольник — хранит три точки, составляющие треугольный полигон

Матрица 4х4 — хранит матрицу размером 4 на 4

Интерфейс — используются библиотечные классы для реализации и предоставления доступа к интерфейсу.

**3. Технологический раздел**

**3.1** **Выбор и обоснование языка программирования и среды разработки**

В качестве языка для разработки редактора был выбран C++ так как:

* Этот язык был в программе курса «Объектно-ориентированное программирование», вследствие чего я его достаточно хорошо знаю для того, чтобы реализовать программный продукт.
* Этот язык был использован мною при прохождении программы курса по компьютерной графике, вследствие чего мною уже были реализованы многие алгоритмы компьютерной графики на этом языке.
* Данный язык является объектно ориентированным, что позволяет:
  + Использовать все инструменты, связанные с парадигмой объектно-ориентированного программирования
  + Предоставлять различные элементы программы в виде объектов классов, что позволит легко организовать взаимодействие между ними, что позволит создавать более качественный код и повышать читаемость кода.
  + Создать приложение, которое можно легко модифицировать, используя механизмы полиморфизма и наследования.

В качестве языка для разработки алгоритма машинного обучения был выбран Python 3, так как:

* Этот язык был в программе курса «Программирование на языке Python», вследствие чего я его достаточно хорошо знаю для того, чтобы реализовать программный продукт.
* Данный язык является объектно ориентированным, что позволяет:
  + Использовать все инструменты, связанные с парадигмой объектно-ориентированного программирования
  + Предоставлять различные элементы программы в виде объектов классов, что позволит легко организовать взаимодействие между ними, что позволит создавать более качественный код и повышать читаемость кода.
  + Создать приложение, которое можно легко модифицировать, используя механизмы полиморфизма и наследования.
* Данный язык является основой для множества библиотек, использование которых позволит добиться более качественного результата работы алгоритма.

В качестве среды разработки редактора была выбрана среда “QT Creator” так как:

* Данная среда является свободно распространяемым программным обеспечением, вследствие чего мы можем использовать её для создания программного продукта.
* В данной среде разработки реализовано множество инструментов, которые позволят существенно облегчить процесс написания и отладки кода.
* Данная среда разработки активно использовалась мною в процессе выполнения работ по курсу «Компьютерная графика»
* Данная среда предоставляет собственные библиотеки для создания интерфейса для приложения, что существенно упрощает его создание.

В качестве среды разработки алгоритма машинного обучения была выбрана среда “Visual Studio Code” так как:

* Данная среда позволяет создавать собственный набор инструментов, который будет использоваться при работе над проектом
* Данная среда поддерживает разработку на языке Python 3
* Данная среда является свободно распространяемым программным обеспечением, вследствие чего мы можем использовать её для создания программного продукта.
* Данная среда разработки является моей основной средой разработки ПО, вследствие чего её использование позволит улучшить качество создаваемого кода.

**3.2 Структура и состав классов**

Класс источника света:

class **DirectionalLight** : public BaseObject

{

public:

**DirectionalLight**();

Vector3 direction;

QColor color;

float power = 1.0f;

};

Класс базового объекта сцены:

class **BaseObject**

{

public:

**BaseObject**();

Vector3 position\_ = {0, 0, 0};

Vector3 rotation\_ = {0, 0, 0};

Vector3 scale\_ = {0, 0, 0};

virtual void ***rotate***(Vector3 delta);

virtual void ***move***(Vector3 delta);

virtual void ***scale***(Vector3 koef);

protected:

Matrix **formMoveMatrix**(Vector3 &deltaCoords);

Matrix **formRotateMatrix**(Vector3 &rotationAnlges);

Matrix **formScaleMatrix**(Vector3 &scaleKoef);

};

Класс модели:

class **Mesh** : BaseObject

{

public:

**Mesh**(); //v

vector<Triangle> tris;

QColor wireframeColor = Qt::white;

void ***rotate***(Vector3 delta) override;

void ***move***(Vector3 delta) override;

void ***scale***(Vector3 koef) override;

};

Класс камеры:

class **Camera** : public BaseObject

{

public:

**Camera**();

Vector3 up\_ = {0, 1, 0};

Vector3 lookDir\_ = {0, 0, 1};

Vector3 right\_ = {1, 0, 0};

Vector3 localRotation\_ = {0, 0, 0};

void ***rotate***(Vector3 delta) override;

void ***move***(Vector3 delta) override;

void **localRotation**(Vector3 delta);

Vector3 **formVectorToPosition**(Vector3 position);

Vector3 **formVectorToPositionNormalized**(Vector3 position);

Matrix **formViewMatrix**();

void **pointAtMatrix**(Vector3 &pos, Vector3 &target, Vector3 &up, Matrix &result);

protected:

void ***scale***(Vector3 koef) override;

};

Класс матрицы:

class **Matrix**

{

public:

**Matrix**(); //v

**Matrix**(Matrix &matrix); //v

**Matrix**(float data[4][4]); //v

void operator= (const Matrix &matrix);

Matrix **dot**(Matrix matrix);

void **identity**();

void **sum**(Matrix &matrix);

void **sum**(float num);

void **sub**(Matrix &matrix);

void **sub**(float num);

void **mult**(Matrix &matrix);

void **mult**(float num);

void **div**(Matrix &matrix);

void **div**(float num);

void **neg**();

Matrix **inverse\_4x4**();

// Upper left square

float **determinant\_2x2**(); //v

float **determinant\_2x2**(float matrix[2][2]); //v

float **determinant\_2x2**(float a, float b, float c, float d); //v

float **determinant\_3x3**(); //v

float **determinant\_3x3**(float matrix[3][3]); //v

float **determinant\_4x4**(); //v

float **determinant\_4x4**(float matrix[4][4]); //v

float data\_[4][4]; //v

};

Класс вектора:

class **Vector3**

{

public:

**Vector3**();

**Vector3**(const Vector3 &vector);

**Vector3**(float x, float y, float z);

**Vector3**(float vector[3]);

float **magnitude**();

float **distance**(Vector3 point);

Vector3 **normalized**();

void **normalize**();

void **vdot**(Matrix &matrix);

void **vdot**(Matrix &matrix, Vector3 &result);

Vector3 operator+ (Vector3 &vector);

Vector3 operator- (Vector3 &vector);

Vector3 operator\* (Vector3 &vector);

Vector3 operator/ (Vector3 &vector);

Vector3 operator+ (float num);

Vector3 operator- (float num);

Vector3 operator\* (float num);

Vector3 operator/ (float num);

void operator= (const Vector3 &vector);

void operator+= (Vector3 vector);

void operator-= (Vector3 vector);

void operator\*= (Vector3 vector);

void operator/= (Vector3 vector);

void operator+= (float num);

void operator-= (float num);

void operator\*= (float num);

void operator/= (float num);

void **copy**(float vector[3]);

void **copy**(Vector3 vector);

float **scalar**(Vector3 vector);

Vector3 **vector**(Vector3 vector);

float x\_;

float y\_;

float z\_;

};

Класс точки:

class **Point3**

{

public:

**Point3**();

float x;

float y;

float z;

};

Класс работы со сценой:

struct **RenderParams**

{

float fAspectRatio;

float fNear;

float fFar;

float fFov;

};

class **SceneManager**

{

public:

**SceneManager**();

**SceneManager**(RenderParams params); //v

float pixelScaleRatio = 100;

Camera mainCamera;

RenderParams renderParams\_;

MeshLoader meshLoader\_;

//DEBUG

Mesh mesh;

void **SetRenderParams**(RenderParams params);

void **renderGizmos**(CanvasLabel &RenderScreen);

void **render**(CanvasLabel &RenderScreen, Mesh &mesh);

protected:

Matrix projMatrix\_;

void **formProjMatrix**(); //v

};

Класс треугольного полигона:

class **Triangle**

{

public:

**Triangle**();

**Triangle**(const Triangle &triangle);

**Triangle**(float points[3][3]);

**Triangle**(Vector3 a, Vector3 b, Vector3 c);

Vector3 **getNormal**();

Vector3 vertices[3];

};

**Заключение**

Во время выполнения поставленной задачи были рассмотрены и проанализированы основные алгоритмы удаления невидимых линий, построения теней, методы закрашивания. Были проанализированы их достоинства и недостатки и были выбраны алгоритмы, наиболее подходящие для решения поставленной задачи. Был предложен способ позволяющий преобразовать изображение в трёхмерную модель, и разработан алгоритм для решения этой задачи. Были выбраны технические средства для реализации программного продукта.

Разработанный в ходе практики программный продукт позволяет синтезировать динамическую сцену с одним трёхмерным объектом при помощи алгоритмов компьютерной графики. Программа позволяет пользователю перемещаться по сцене и изменять положение и вращение объекта.

В ходе решения поставленной задачи были изучены возможности Qt Creator, получены знания в области компьютерной графики и опыт программирования на языке C++.

**Список литературы**

[1] Ю. М. Баяковский, В. А. Галактионов, О некоторых фундаментальных проблемах компьютерной графики, ИТиВС, 2004, выпуск 4.

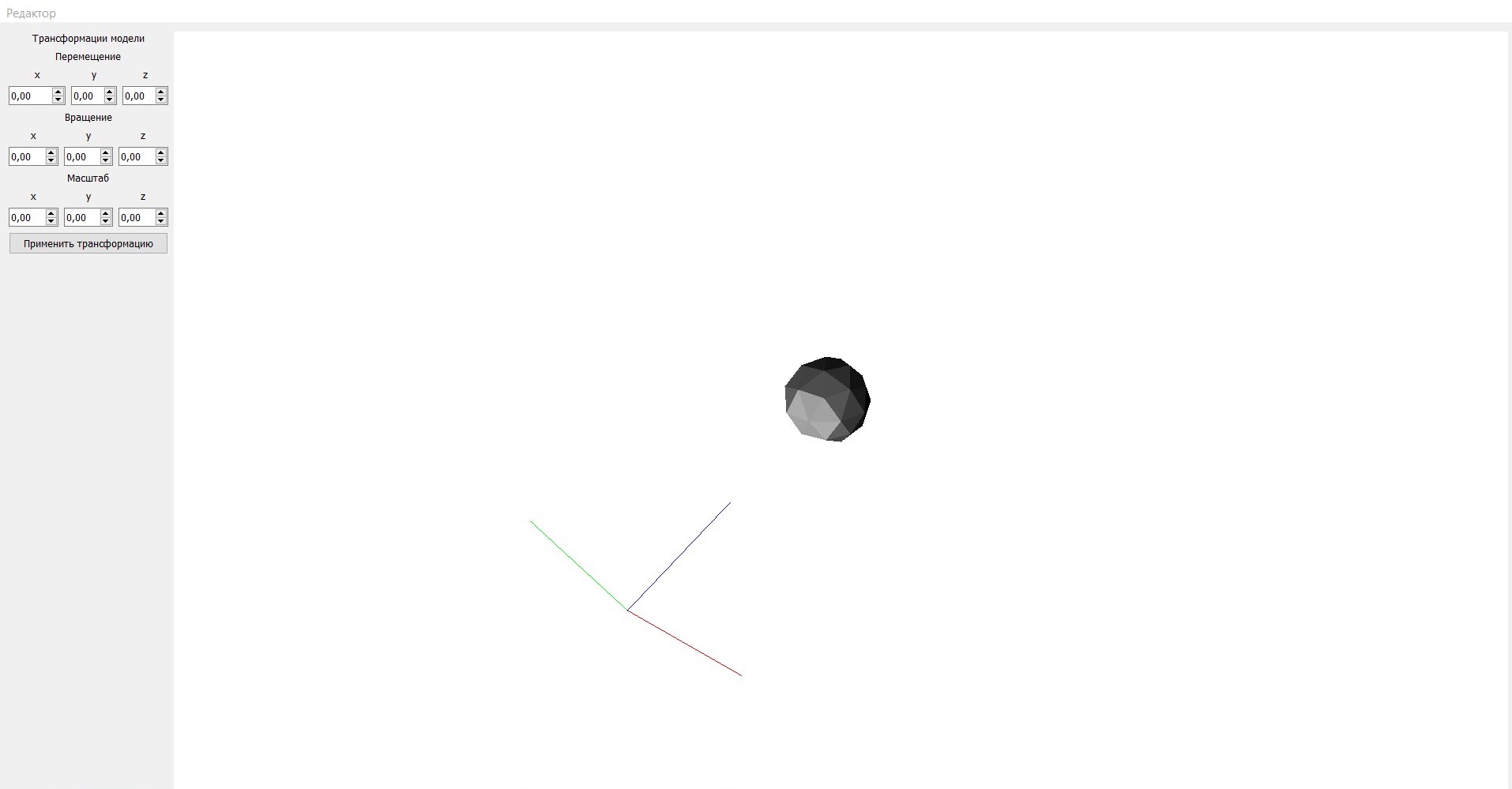
[2] Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. Differentiable monte carlo ray

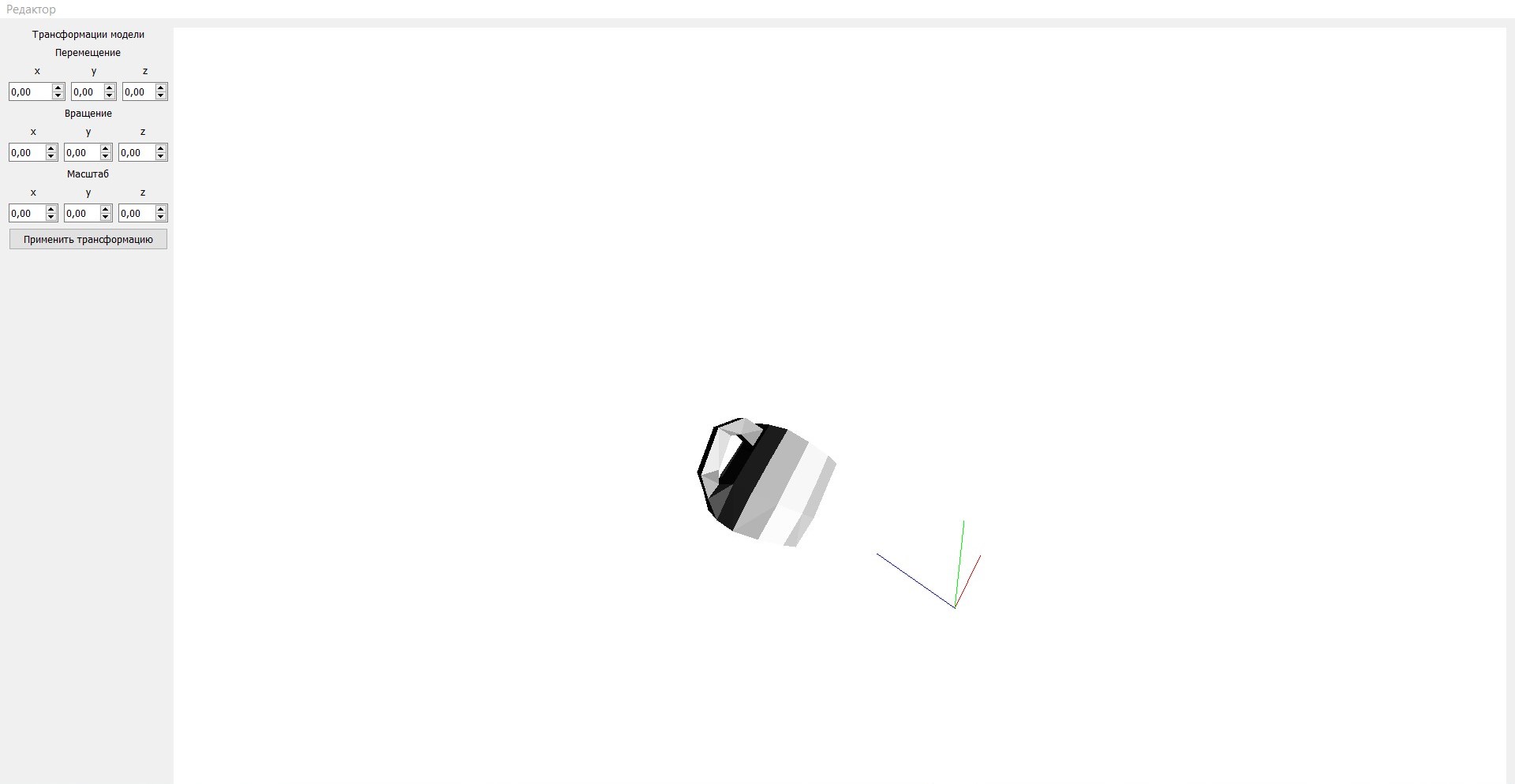
tracing through edge sampling. In SIGGRAPH Asia 2018 Technical Papers, page 222. ACM,

2018.

[3] Н. А. Сторчак, В. И. Гегучадзе, А. В. Синьков, Моделирование трёхмерных объектов в среде Компас 3D, 2006, с 5.

**Приложения**

*Рисунок 1. Визуализация сцены с изосферой.*

*Рисунок 2. Визуализация сцены с чашкой.*