|  |  |
| --- | --- |
| 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 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Жабин Дмитрий Владимирович\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*фамилия, имя, отчество*

Группа \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ИУ7-44Б\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Тип практики \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_технологическая\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Название предприятия \_\_\_\_\_\_МГТУ им. Н. Э. Баумана, каф. ИУ7\_\_\_\_\_\_\_\_\_\_\_\_

Студент **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_**Жабин Д.В.\_**\_\_\_\_**

*подпись, дата фамилия, и.о.*

Руководитель практики **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_**Куров А.В.**\_\_\_\_\_\_\_**

*подпись, дата фамилия, и.о.*

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

*Москва, 2021 г.*

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)

Кафедра «Программное обеспечение ЭВМ и информационные технологии» (ИУ7)

**ЗАДАНИЕ**

**на прохождение производственной практики**

на предприятии \_\_\_\_\_\_\_\_\_\_\_\_\_МГТУ им. Н.Э. Баумана, каф. ИУ7\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Жабин Дмитрий Владимирович ИУ7-44Б\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(фамилия, имя, отчество; инициалы; индекс группы)

Во время прохождения производственной практики студент должен:

1. Спроектировать программу реализации и исследования алгоритма z-буфера удаления невидимых поверхностей.

2. Предоставить пользователю возможность добавлять объекты в сцену (загружать полигональные модели из файла).

3. Предоставить пользователю возможность изменять положение источника света.

4. Модифицировать алгоритм так, чтобы он включал построение теней, отбрасываемых объектами.

Дата выдачи задания « \_\_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_ г.

Руководитель практики от кафедры \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_Куров А.В.\_\_\_\_\_

(подпись, дата) (Фамилия И.О.)

Студент **\_\_**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_Жабин Д.В.\_\_\_\_\_

(подпись, дата) (Фамилия И.О.)

Оглавление

[Введение 4](#_Toc14169990)

[1. Аналитическая часть 5](#_Toc14169991)

[1.1 Формализация объектов синтезируемой сцены 5](#_Toc14169992)

[1.2 Анализ алгоритмов удаления невидимых линий и поверхностей 6](#_Toc14169993)

[1.3 Анализ методов закрашивания 13](#_Toc14169994)

[1.4 Анализ алгоритмов построения теней 14](#_Toc14169995)

[2. Конструкторская часть 15](#_Toc14169997)

[2.1 Общий алгоритм решения задачи 15](#_Toc14169998)

[2.2 Алгоритм Z-буфера 15](#_Toc14169999)

[2.3 Простой метод освещения 16](#_Toc14170000)

[2.4 Выбор используемых типов и структур данных 16](#_Toc14170002)

[3. Технологическая часть 17](#_Toc14170003)

[3.1 Выбор и обоснование языка программирования и среды разработки 17](#_Toc14170004)

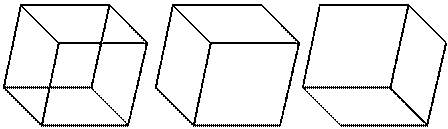
[Заключение 18](#_Toc14170008)

[Список использованной литературы 19](#_Toc14170009)

# Введение

Одной из основных задач компьютерной графики является визуализация трёхмерных сцен. Подобные задачи возникают в системах автоматизированного проектирования, пакетах моделирования физических процессов, средствах компьютерной анимации и виртуальной реальности. При отображении трёхмерной сцены на экране некоторые из объектов сцены могут заслонить другие объекты. Заслонённые части объектов невидимы и не должны рисоваться, или должны рисоваться иначе, чем видимые части, например, пунктиром. Если этого не делать, то изображение будет выглядеть неправильно. Такие задачи решают с помощью алгоритмов удаления невидимых линий и поверхностей. Невидимые линии удаляются при отображении сцены в каркасном виде (алгоритм выделяет части ребер объекта, которые заслонены и удаляет их). При отображении объектов с помощью закрашенных поверхностей удаляются их невидимые части.

Необходимость удаления невидимых линий, ребер, поверхностей или объемов проиллюстрирована на рис.1.



а                          б                              в

Рис.1. Необходимость удаления невидимых линий

На рис. 1.а приведен типичный каркасный чертеж куба. Каркасный чертеж представляет трехмерный объект в виде штрихового изображения его ребер. Рис. 1.а можно интерпретировать двояко: как вид куба сверху слева или снизу справа. Удаление тех линий или поверхностей, которые невидимы с соответствующей точки зрения, позволяют избавиться от неоднозначности. Результаты показаны на рис. 1.б и 1.в.

В данной работе необходимо реализовать алгоритм удаления невидимых линий и поверхностей и произвести анализ его производительности.

# Аналитическая часть

## 1.1 Формализация объектов синтезируемой сцены

Сцена состоит из:

* Источника света – представляет собой материальную точку, из которой исходят лучи света во все стороны (в частном случае, когда источник расположен в бесконечности, имеет направленность).

Положение задается трехмерными координатами, цвет свечения описывается через RGB параметры.

* Фигур – многогранников, добавленных в сцену пользователем (описываются в файле)

Фигуры наилучшим образом описываются через поверхностные модели.

Поверхностную модель можно задать несколькими способами:

**Параметрическим представлением** – для получения поверхности нужно вычислять функцию, зависящую от параметра. В сцене нет поверхностей вращения, поэтому использование параметрического представления будет затруднительно и нецелесообразно.

**Полигональной сеткой** – совокупностью вершин, ребер и граней, которые определяют форму объекта.

* **Вершинное представление** описывает объект как множество вершин, соединённых с другими вершинами. Это простейшее представление, но оно не широко используемое, так как информация о гранях и рёбрах не выражена явно. Поэтому нужно обойти все данные, чтобы сгенерировать список граней для рендеринга. Кроме того, нелегко выполняются операции на рёбрах и гранях.
* **Список граней** представляет объект как множество граней и множество вершин. В отличие от вершинного представления, и грани и вершины явно представлены, так что нахождение соседних граней и вершин постоянно по времени. Моделирование требует лёгкого обхода всех структур. Однако, рёбра не заданы явно, так что поиск всё ещё нужен, чтобы найти все грани, окружающие заданную грань. Другие динамические операции, такие как разрыв или объединение грани, также сложны со списком граней.
* **«Крылатое» представление**: в нём каждая точка ребра указывает на две вершины, две грани и четыре (по часовой стрелке и против часовой) ребра, которые её касаются. Крылатое представление позволяет обойти поверхность за постоянное время, но у него большие требования по памяти хранения. Хорошо подходит для динамической геометрии.

В качестве способа хранения объектов моей сцены был выбран список граней, так как это представление обеспечивает удобную работу с каждой гранью в отдельности, что необходимо при реализации алгоритма, использующего z-буфер.

## 1.2 Анализ алгоритмов удаления невидимых линий и поверхностей

Алгоритмы удаления невидимых линий или поверхностей можно классифицировать по способу выбора системы координат или пространства, в котором они работают. Выделяют три класса алгоритмов удаления невидимых линий или поверхностей:

* Алгоритмы, работающие в объектном пространстве.
* Алгоритмы, работающие в пространстве изображения (экрана).
* Алгоритмы, формирующие список приоритетов.

Алгоритмы, работающие в объектном пространстве, имеют дело с физической системой координат, в которой описаны эти объекты. При этом получаются весьма точные результаты, ограниченные лишь точностью вычислений. Полученные изображения можно свободно увеличивать во много раз. Алгоритмы, работающие в объектном пространстве, особенно полезны в тех приложениях, где необходима высокая точность.

Алгоритмы же, работающие в пространстве изображения, имеют дело с системой координат того экрана, на котором объекты визуализируются. При этом точность вычислений ограничена разрешающей способностью экрана. Обычно разрешение экрана бывает довольно низким. Результаты, полученные в пространстве изображения, а затем увеличенные во много раз, не будут соответствовать исходной сцене. Например, могут не совпасть концы отрезков.

Алгоритмы, формирующие список приоритетов, работают попеременно в обеих упомянутых системах координат.

Объем вычислений для любого алгоритма, работающего в объектном пространстве и сравнивающего каждый объект сцены со всеми остальными объектами этой сцены, растет теоретически, как квадрат числа объектов (*n*2). Аналогично, объем вычислений любого алгоритма, работающего в пространстве изображения и сравнивающего каждый объект сцены с позициями всех пикселов в системе координат экрана, растет теоретически, как *nN*. Здесь *n*обозначает количество объектов (тел, плоскостей или ребер) в сцене, а *N* — число пикселов. Теоретически трудоемкость алгоритмов, работающих в объектном пространстве, меньше трудоемкости алгоритмов, работающих в пространстве изображения, при *n < N*. Однако на практике это не так. Дело в том, что алгоритмы, работающие в пространстве изображения, более эффективны потому, что для них легче воспользоваться преимуществом когерентности при растровой реализации.

**Алгоритм определения видимых поверхностей путем трассировки лучей**

В данном алгоритме для каждого пикселя изображения находится ближайшая к нему грань. Для чего через этот пиксель выпускается луч, находятся все его пересечения с гранями и среди них выбирается ближайшее к наблюдателю пересечение. Данный алгоритм имеет решение проблемы максимально близкое к природе света, что позволяет с помощью данного алгоритма также отрисовывать тени, отражения, учитывать глобальную модель освещения.

**Алгоритм плавающего горизонта**

Алгоритм плавающего горизонта применяется для удаления невидимых линий при изображении трехмерных плоскостей, задаваемых функциями вида

𝐹(𝑥,𝑦,𝑧) = 0. Идея алгоритма заключается в анализе пересечения данной функции с режущими плоскостями вида 𝑧 = 𝑐𝑜𝑛𝑠𝑡. Пересечения, представляющие собой кривые, рассматриваются в порядке удаления от наблюдателя. Для каждой режущей плоскости для определенного значения 𝑥 и соответствующего значения 𝑦 кривой сравнивается со значениями 𝑦 для всех предыдущих кривых при том же значении 𝑥. Если точка выше или ниже всех других точек, тогда она видима. Таким образом алгоритм работает в пространстве изображения.

Данный алгоритм можно модифицировать для построения полигональных моделей, но тогда будет требоваться строгая сортировка полигонов по удалению от наблюдателя, что не в каждой сцене достижимо.

**Алгоритм Робертса**

Робертсу удалось одним из первых решить проблему удаления невидимых линий и поверхностей. Данный алгоритм работает в пространстве объектов сцены. Алгоритм удаляет в первую очередь нелицевые грани тел. Затем удаляются перекрытия одних тел другими.

Данный алгоритм работает с объектами как с набором пересекающих плоскостей, из-за чего возможна работа только с выпуклыми телами. Для построения полигональных моделей, необходимо предварительное разбиение тел на выпуклые тела.

Идеей первого этапа является анализ по какую сторону от плоскости находится наблюдатель, если наблюдатель находится по отрицательную сторону плоскости, то грань, образуемая этой плоскостью невидима. Затем требуется найти отрезки, которые перекрываются другими телами. Для этого каждое остававшееся ребро необходимо сравнить с другими телами сцены. Для ускорения процесса рекомендуется отсортировать ребра по удалению от наблюдателя.

**Алгоритм Варнока**

Алгоритм Варнока основывается на рекурсивном разбиении экрана. Алгоритм работает в пространстве изображения, которое разбивается на окна, пока задача по определению, какой сегмент нужно отрисовать в данном окне, не станет тривиальной. Таким образом разбиение изображение возможно до одного пикселя. Варнок изначально предложил делить окна на одинаковые четыре подокна, но данный алгоритм варьируется способами разбиения изображения на подокна. Например, Вейлером и Азертоном было предложено разбивать окна по границам многоугольников.

Данный алгоритм хорошо работает для простых сцен с малым количеством тел. Но с ростом сложности сцены разбиваться сцена может вплоть до одного пикселя, что плохо сказывается на производительности.

**Алгоритм, использующий список приоритетов**

Для работы данного алгоритма необходимо предварительно отсортировать по глубине грани тел. Так как элементы, расположенные ближе к наблюдателю, перекрывают элементы, находящиеся за ним, проблема удаления невидимых поверхностей решается тривиально - путем поочередного изображения элементов в порядке приближения к наблюдателю. Для простых элементов сцены, таких как многоугольники, этот алгоритм иногда называют алгоритмом художника, так как он похож на то, как художник пишет картину.

Основной проблемой алгоритма является неоднозначность процесса сортировки граней. Проблема заключается в том, что грань может принимать несколько значений на одной оси. Также данный алгоритм без предварительного разбиения не может решить проблему с многоугольниками, циклически перекрывающихся друг другом.

## Алгоритм, использующий z–буфер

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 *х* и *у* наибольшего значения функции *z (х, у).*

Главное преимущество алгоритма – его простота. Кроме того, этот алгоритм решает задачу об удалении невидимых поверхностей и делает тривиальной визуализацию пересечений сложных поверхностей. Сцены могут быть любой сложности. Поскольку габариты пространства изображения фиксированы, оценка вычислительной трудоемкости алгоритма не более чем линейна. Поскольку элементы сцены или картинки можно заносить в буфер кадра или в *z*-буфер в произвольном порядке, их не нужно предварительно сортировать по приоритету глубины. Поэтому экономится вычислительное время, затрачиваемое на сортировку по глубине.

**Формальное описание алгоритма z-буфера** таково:

1.   Заполнить буфер кадра фоновым значением интенсивности или цвета.

2.   Заполнить *z*-буфер минимальным значением *z*.

3.   Преобразовать каждый многоугольник в растровую форму в произвольном порядке.

4.   Для каждого *Пиксел*(*x,y*) в многоугольнике вычислить его глубину *z*(*x,y*).

5.   Сравнить глубину *z*(*х,у*) со значением *Zбуфер*(*х,у*), хранящимся в *z*-буфере в этой же позиции.

6. Если *z*(*х,у*) > *Zбуфер* (*х,у*), то записать атрибут этого многоугольника (интенсивность, цвет и т. п.) в буфер кадра и заменить *Zбуфер*(*х,у*) на *z*(*х,у*). В противном случае никаких действий не производить.

Если известно уравнение плоскости, несущей каждый многоугольник, то вычисление глубины каждого пиксела на сканирующей строке можно проделать пошагово. Грань при этом рисуется последовательно (строка за строкой).

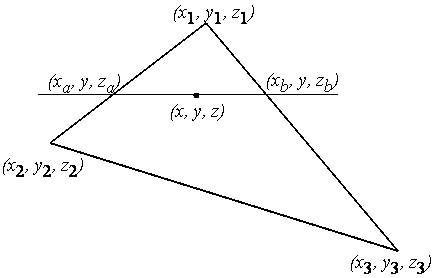


Рис. 2. Сканирующая строка по грани

Для рисунка *y* меняется от *y*1 до *y*2 и далее до *y*3, при этом для каждой строки определяется *xa, za, xb, zb*:

|  |  |  |
| --- | --- | --- |
| *xa*= *x*1+ (*x*2- *x*1) | http://compgraph.tpu.ru/zbuffer.files/image003.gif | ; |
| *xb*= *x*1+ (*x*3- *x*1) | http://compgraph.tpu.ru/zbuffer.files/image005.gif | ; |
| *za*= *z*1+ (*z*2- z1) | http://compgraph.tpu.ru/zbuffer.files/image003.gif | ; |
| *zb*= *z*1+ (*z*3- z1) | http://compgraph.tpu.ru/zbuffer.files/image005.gif | . |

На сканирующей строке *x* меняется от *xa* до *xb* и для каждой точки строки определяется глубина *z*:

|  |  |  |
| --- | --- | --- |
| *z*= *za*+ (*zb*- *za*) | http://compgraph.tpu.ru/zbuffer.files/image009.gif |  |

Реализация алгоритма вдоль сканирующей строки позволяет совместить алгоритм *z*-буфера с алгоритмами растровой развертки ребер и алгоритмами закраски грани.

Пример работы алгоритма показан на рис. 3.

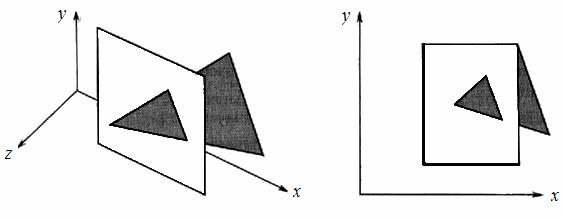
**

Рис. 3. Протыкающий треугольник

**Вывод**

Алгоритм, использующий z-буфер, отлично подходит для реализации поставленной задачи, так как он позволяет быстро строить произвольные полигональные модели с использование однотонной закраски. Данный алгоритм требует модификации для построения теней, отбрасываемых объектами. Размер синтезируемого изображения сравнительно мал, поэтому затраты по памяти, необходимые для хранения информации о каждом пикселе, незначительны для современных компьютеров.

## 

## 1.3 Анализ методов закрашивания

**Простая закраска**

Вся грань закрашивается одним уровнем интенсивности, который высчитывается по закону Ламберта.

Этот метод крайне прост в реализации и совершенно не требователен к ресурсам. Однако плохо подходит для тел вращения, плохо учитывает отраженный свет.

Этот метод хорошо подходит для моей задачи, так как вся работа ведется с плоскими гранями, тел вращения нет.

**Закраска по Гуро**

Основа закраски по Гуро – билинейная интерполяция интенсивностей, за счет которой устраняется дискретность изменения интенсивности и создается иллюзия гладкой криволинейной поверхности. Хорошо сочетается с диффузным отражением.

**Закраска по Фонгу**

Основа закраски по Фонгу – билинейная интерполяция векторов нормалей. Достигается лучшая локальная аппроксимация кривизны поверхности. Изображение выходит более реалистичным, зеркальные блики выглядят правдоподобнее, чем в методе закраски по Гуро.

Однако по сравнению с методом Гуро, закраска по Фонгу требует больших вычислительных затрат, так как интерполируются значения векторов нормалей, на основе которых потом вычисляется интенсивность.

**Вывод**

Фигуры сцены состоят из плоскостей, поэтому закраска по Фонгу и Гуро будет скорее мешать: ребра многогранников будут сглажены. Поэтому лучше всего использовать простую закраску.

## 

## 1.4 Анализ алгоритмов построения теней

При трассировке лучей тени получаются без дополнительных вычислений.  
Пиксел затенен, когда луч попадает на объект и позже не попадает ни в объект, ни в источник света.

В анализе алгоритмов трассировка лучей не была выбрана в качестве алгоритма синтеза сцены, поэтому тени нужно вычислять отдельно.

Один из способов нахождения теней – вычисление проекций тел.

Можно использовать метод теневого буфера, в котором предполагается, что освещены только те точки, которые видны из положения источника. В свою очередь, алгоритм Z-буфера позволяет быстро находить видимость объектов сцены.

# 2. Конструкторская часть

## 2.1 Общий алгоритм решения задачи

1. Задать объекты сцены

2. Задать положение источника света и положение наблюдателя

3. Для каждого полигона высчитать нормаль и интенсивность цвета

4. Используя алгоритм Z буфера совместно с теневым буфером, получить изображение сцены

## 2.2 Алгоритм Z-буфера

1. Всем элементам буфера кадра присвоить фоновое значение
2. Инициализировать Z буфер минимальными значениями глубины
3. Выполнить растровую развертку каждого многоугольника сцены:
   1. Для каждого пикселя, связанного с многоугольником вычислить его глубину z(x, y)
   2. Сравнить глубину пикселя со значением, хранимым в Z буфере.

Если z(x, y) > zбуф(x, y), то zбуф(x,y) = z(x,y), цвет(x, y) = цветПикселя.

1. Отобразить результат

**Построение теней с использованием алгоритма z-буфера**

При построении теней с использованием алгоритма z-буфера выполняется два прохода: один - относительно источника света, другой – относительно наблюдателя. Для этого выделяется отдельный "теневой" z-буфер. Первый проход необходим для того, чтобы определить, какие точки видны со стороны источника света. При втором проходе сцена визуализируется из положения наблюдателя с учетом того, что точки, которые оказались невидимыми со стороны источника света, находятся в тени. Таким образом, алгоритм работает в два этапа. При первом проходе сцена рассчитывается при совмещении точки наблюдения с положением источника света. Значения глубины пикселей для данного вида заносятся в "теневой" z-буфер, а значения интенсивности не учитываются. При втором проходе сцена строится из точки, в которой на самом деле находится наблюдатель. При сканировании каждой поверхности значение ее глубины каждого ее пиксела сравнивается со значением глубины в z-буфере. Если поверхность видима, то необходимо проверить, видима ли данная точка со стороны источника света. Для этого координаты точки x, y, z из вида наблюдателя линейно преобразуются в координаты x', y', z' на виде из источника света. Проверка на видимость осуществляется сравнением значения, которое хранится в "теневом" z-буфере для координат x', y', и значения z'. Если точка невидима для источника света (значение в "теневом" z-буфере больше значения z'), значит она находится в тени и ее свечение вычисляется с учетом затенения. Если же точка видима из положения источника света, то она изображается без изменений.

## 2.3 Простой метод освещения

В простом методе освещения интенсивность рассчитывается по закону Ламберта:

I = I0\*cos(α), где

I – результирующая интенсивность света в точке

I0 – интенсивность источника

α – угол между нормалью к поверхности и вектором направления света

## 

## 2.4 Выбор используемых типов и структур данных

* Источник света – задается расположением и направленностью света.
* Объекты сцены – задаются множествами вершин и граней.
* Математические абстракции:
  + Точка – хранит координаты x, y, z
  + Многоугольник – хранит вершины, нормаль, цвет

# 3. Технологическая часть

## 3.1 Выбор языка программирования и среды разработки

В качестве языка программирования был выбран C++.

Данный выбор обоснован высокой скоростью работы языка, поддержкой объектно-ориентированного подхода программирования и строгой типизацией.

Было принято решение использовать Qt Framework, который предоставляет большой набор базовых компонентов, необходимых для решения поставленной задачи. Кроме того, я ознакомлен со средой разработки Qt Creator, которая облегчит процесс написания и отладки кода.

# 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, выбраны наиболее подходящие для решения поставленной задачи.

Спроектированный программный продукт визуализирует трехмерное изображение при помощи алгоритмов компьютерной графики.

В ходе выполнения поставленной задачи были получены знания в области компьютерной графики.

# Список использованной литературы

1. Методы представления дискретных данных [Электронный ресурс]. – Режим доступа:

<https://www.graphicon.ru/oldgr/ru/library/multires_rep/index.html> (дата обращения 17.07.21)

1. Полигональная сетка [Электронный ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D0%B8%D0%B3%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D0%B5%D1%82%D0%BA%D0%B0> (дата обращения 17.07.21)
2. Е. А. Снижко. Компьютерная геометрия и графика [Текст], 2005. - 17 с.
3. Qt Widgets 5.15.5 [Электронный ресурс]. — Режим доступа: https://doc.qt.io/qt-5/qtwidgets-index.html (дата обращения: 17.07.2021).