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:

«Программа моделирования трехмерного изображения баскетбольного мяча»

Студент ИУ7-54Б _____ **А.М. Турчанинов**
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель _____ **Н.Б. Толпинская**
(Подпись, дата) (И.О.Фамилия)

2022 г.

УТВЕРЖДАЮ

Заведующий кафедрой ИУ7
(Индекс)
_____ И.В.Рудаков
(И.О.Фамилия)
« __ » _____ 2022 г.

З А Д А Н И Е
на выполнение курсовой работы

по дисциплине Компьютерная графика
Программа моделирования трехмерного изображения баскетбольного мяча
(Тема курсового проекта)

Студент Турчанинов А.М. гр. ИУ7-54
(Фамилия, инициалы, индекс группы)

График выполнения проекта: 25% к 4 нед., 50% к 7 нед., 75% к 11нед., 100% к 14 нед.

1. Техническое задание

Разработать программу моделирования трехмерного изображения мяча, интерфейс которой должен предоставлять пользователю возможность рассмотреть мяч со всех сторон при помощи изменения положения камеры, смещения и поворота мяча.

2. Оформление курсовой работы

2.1. Расчетно-пояснительная записка на 25-30 листах формата А4.

Расчетно-пояснительная записка должна содержать постановку задачи, введения, аналитическую часть, конструкторскую часть, технологическую часть, экспериментально-исследовательский раздел, заключение, список литературы, приложения.

2.2. Перечень графического материала (плакаты, схемы, чертежи и т.п.). На защиту проекта должна быть представлена презентация, состоящая из 15-20 слайдов. На слайдах должны быть отражены: постановка задачи, использованные методы и алгоритмы, расчетные соотношения, структура комплекса программ, диаграмма классов, интерфейс, характеристики разработанного ПО, результаты проведенных исследований.

Дата выдачи задания « 25 » мая 2022 г.

Руководитель курсовой работы _____ Н.Б. Толпинская
(Подпись, дата) (И.О.Фамилия)

Студент _____ А.М.Турчанинов
(Подпись,дата) (И.О.Фамилия)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В наше время компьютерная графика имеет достаточно широкий охват применения во всех отраслях нашей жизни: в кино, в компьютерных играх, в рекламе и дизайне, в обучающих программах. Перед специалистами, создающими трехмерные изображения появляется множество сложностей таких как: преломление, отражение и рассеивание света. Знания компьютерной графики никогда не станут лишним багажом, они только станут преимуществом на собеседованиях с работодателями, помогут самостоятельно создавать персонажей для компьютерных игр, генерировать спецэффекты для фильмов. Необходимость широкого использования графических программных средств стала особенно ощутимой в связи с развитием Интернета. Способность компьютерной графики быть многозадачной, необычной и символичной. Применение графики в учебных системах не только позволяет увеличить скорость передачи информации и повысить уровень её понимания, но и способствует развитию образного мышления.

Мною была выбрана тема, связанная с моделированием баскетбольного мяча.

Для исполнителя важно исключить возникновение правок на поздних стадиях выполнения работы. Поэтому важно представить реалистичное изображение на этапе предварительного показа. Для построения такого изображения, которое позволит понять пользователю концепцию работы, потребуется учитывать невидимость ребер объектов сцены по отношению к наблюдателю и освещение отдельных участков рабочей плоскости.

Целью курсовой работы является реализация программного обеспечения для моделирования баскетбольного мяча. Для достижения поставленной цели необходимо выполнить следующие задачи.

- Описать список доступных к размещению на сцене моделей, формализовать эти модели;
- Выбрать существующие алгоритмы компьютерной графики для визуализации сцены и объектов на ней;
- Выбрать язык программирования и среду разработки.;

– Реализовать выбранные алгоритмы визуализации;

– Реализовать программное обеспечение для визуализации и редактирования сцены.

1 Аналитическая часть

1.1 Описание и формализация объектов сцены

Сцена состоит из следующих объектов.

– Область действий -– куб, внутри которого располагаются модель мяча, два источника света и зритель. Размер куба постоянный, зритель может перемещаться по области и рассматривать мяч с разных сторон;

– Объекты сцены — модели, расположенные внутри области действий. Каждая модель представляет собой набор граней, описываемых точками в пространстве, которые соединены ребрами. Все доступные модели определены заранее, в программе не предусмотрена возможность добавления новых или изменения старых моделей;

Список доступных объектов сцены

– Мяч. Разноцветный объект шаровидной формы. На него с двух противоположных углов куба светят два источника света. Мяч можно рассмотреть со всех сторон при помощи перемещения взгляда наблюдателя;

– Наблюдатель. Смотрит на мяч, может перемещаться по трем осям, таким образом рассматривая мяч с разных сторон;

– Источники света — материальные точки пространства. Принимает ортогональную проекцию визуализируемой сцены из своего положения с некоторым ограниченным обзором. В зависимости от расположения источника и направления распространения лучей света, определяется тень от объектов, расположенных в области действий. Положение источника света зафиксировано.

1.2 Анализ и выбор формы задания трехмерных моделей

Отображением формы и размеров объектов являются модели.
Обычно используются три формы задания моделей:

- Каркасная (проволочная) модель. Одна из простейших форм задания модели, так как мы храним только информацию о вершинах и ребрах нашего объекта. Недостаток – модель не всегда точно передает представление объекта;
- Поверхностная модель. Тип модели, часто используемый в компьютерной графике. Поверхности задаются разными способами – аналитически или задаются участки поверхности, как поверхности разных видов(использование полигональной аппроксимации). Недостаток – мы не знаем с какой стороны находится материал ;
- Объемная (твердотельная) модель. Главное отличие от поверхностной модели – информация о том, где расположен материал. Достигается это посредством указания направления внутренней нормали. .

Выбор модели

При решении данной задачи подойдут будет использоваться поверхностная модель. Этот выбор обусловлен тем, что каркасные модели могут привести к неправильному восприятию формы объекта, а реализация объемной модели потребует большего количества ресурсов на воспроизведение деталей, не влияющих на качество решения задачи в ее заданной формулировке.

1.3 Анализ способа задания поверхностных моделей

Также необходимо определить каким образом лучше всего задавать поверхностные модели:

– Аналитическим способом. Этот способ задания модели характеризуется описанием модели объекта, которое доступно в неявной форме, то есть для получения визуальных характеристик необходимо дополнительно вычислять некоторую функцию, которая зависит от параметра;

– Полигональной сеткой. Данный способ характеризуется совокупностью вершин, граней и ребер, которые определяют форму многогранного объекта в трехмерной графике.

Для верного выбора стоит перечислить способы хранения информации о сетке:

– Список граней. Объект – это множество граней и множество вершин. В каждую грань входит как минимум 3 вершины;

– "Крылатое"представление. Каждая точка ребра указывает на две вершины, две грани и четыре ребра, которые её касаются;

– Полурёберные сетки. Похоже на "Крылатое"представление, но информация обхода хранится для половины грани;

– Таблица углов. Таблица, в которой хранятся вершины. Обход заданной таблицы неявно задает полигоны. Такое представление более компактно и более производительно для нахождения полигонов, но, в связи с тем, что вершины присутствуют в описании нескольких углов, операции по их изменению медленны;

– Вершинное представление. Хранятся только вершины, указывающие на другие вершины. Простота представления дает возможность проводить над сеткой множество операций.

Решающий фактор при выборе задания модели в проекте – скорость выполнения преобразований над объектами сцены.

Наиболее удобный способ представления для реализации данного программного обеспечения – модель, заданная полигональной сеткой, так как это поможет избежать проблем при описании сложных моделей. Способ хранения полигональной сетки – список граней, так как он представляет явное описание граней, что поможет при реализации алгоритма удаления невидимых ребер и поверхностей. Еще одно преимущество этого способа – эффективное преобразование модели, так как структура включает в себя список вершин.

1.4 Анализ и выбор алгоритма удаления невидимых ребер и поверхностей

Чтобы выбрать правильный алгоритм необходимо определить некоторые свойства, которыми должен обладать выбранный алгоритм для обеспечения реалистичного изображения и оптимальную работу.

Свойства:

– Алгоритм должен быть быстрым и использовать мало памяти;

– Алгоритм должен иметь высокую реалистичность изображения.

Рассмотрим алгоритмы для удаления невидимых ребер и поверхностей.

1.4.1 Алгоритм Робертса

Данный алгоритм работает в объективном пространстве, решая задачу только с выпуклыми телами.

Алгоритм выполняется в три этапа:

Этап подготовки исходных данных

На данном этапе задается информация о телах. Для каждого тела формируется матрица тела V. Размерность матрицы – 4 * n, где n – количество

граней тела. Каждый столбец матрицы представляет собой четыре коэффициента уравнения плоскости ax + by + cz + d = 0, проходящей через очередную грань. Таким образом, матрица тела будет представлена в следующем виде:

$$V = \begin{pmatrix} a_1 & a_2 & \dots & a_n \\ b_1 & b_2 & \dots & b_n \\ c_1 & c_2 & \dots & c_n \\ d_1 & d_2 & \dots & d_n \end{pmatrix}$$

Матрица тела должна быть сформирована корректно, то есть любая точка, расположенная внутри тела, должна располагаться по положительную сторону от каждой грани тела. В случае, если для очередной грани условие не выполняется, соответствующий столбец матрицы надо умножить на -1. Для проверки надо взять точку, расположенную внутри тела. Координаты такой точки можно получить путем усреднения координат всех вершин тела.

Этап удаления ребер, экранируемых самым телом

На данном этапе рассматривается вектор взгляда E = 0, 0, -1, 0. Чтобы определить невидимые грани необходимо умножить вектор E на матрицу тела V. Отрицательные компоненты полученного вектора – невидимые грани.

Этап удаления невидимых ребер, экранируемых другими телами сцены

На данном этапе строим луч, соединяющий точку наблюдения с точкой на ребре, для определения невидимых точек ребра. Точка невидима, если луч на своем пути встречает преграду – рассматриваемое тело. Если тело является преградой, то луч должен пройти через тело. Если луч проходит через тело, то он находится по положительную сторону от каждой грани тела.

Преимущества и недостатки алгоритма Робертса

Преимущества:

– алгоритм работает в объектном пространстве, высокая точность вычислений;

Недостатки:

– все тела сцены должны быть выпуклыми. Эта проблема также усложняет алгоритм, так как становится необходима проверка на выпуклость и разбиению на выпуклые многоугольники;

Данный алгоритм не подходит для решения поставленной задачи из-за высокой сложности реализации как самого алгоритма, так и его модификаций, отсюда низкая производительность.

1.4.2 Алгоритм, использующий Z-буфер

Суть данного алгоритма — использование двух буферов: буфера кадра, в котором хранятся атрибуты каждого пикселя, и Z-буфера, в котором хранится информация о координате Z для каждого пикселя.

Первоначально в Z-буфере находятся минимально возможные значения Z, а в буфере кадра располагаются пиксели, описывающие фон. Каждый многоугольник преобразуется в растровую форму и записывается в буфер кадра.

В процессе подсчета глубины нового пикселя, он сравнивается с тем значением, которое уже лежит в Z-буфере. Если новый пиксель расположен ближе к наблюдателю, чем предыдущий , то он заносится в буфер кадра и происходит корректировка Z-буфера.

Для решения задачи вычисления глубины Z каждый многоугольник описывается уравнением ax + bx+ cz +d = 0. При c = 0 многоугольник для наблюдателя вырождается в линию.

Для некоторой сканирующей строки y = const, поэтому имеется возможность рекуррентно высчитывать z’ для каждого x’ = x + dx:

Получим: z’ = z/c, так как x – x’ = dx = 1.

$$z' - z = -\frac{ax' + d}{c} + \frac{ax + d}{c} = \frac{a(x - x')}{c}$$

При этом стоит отметить, что для невыпуклых многогранников предварительно потребуется удалить нелицевые грани.

Преимущества и недостатки алгоритма, использующего Z-буфер

Преимущества:

– простая реализация;

– оценка трудоемкости линейна;

– экономия вычислительного времени, так как элементы сцены не сортируются.

Недостатки:

– сложная реализация прозрачности;

– большой объем требуемой памяти.

Данный алгоритм не подходит для решения поставленной задачи, так как требует большого объема памяти, что не удовлетворяет требованиям.

1.4.3 Алгоритм обратной трассировки лучей

Суть данного алгоритма заключается в том, что наблюдатель видит объект с помощью испускаемого света, который по законам оптики доходит до наблюдателя некоторым путем. Отслеживать пути лучей от источника к наблюдателю неэффективно с точки зрения вычислений, поэтому лучшим способом будет отслеживание в обратном направлении, то есть от наблюдателя к объекту.

Преимущества:

– высокая реалистичность изображения;

– вычислительная сложность не особо зависит от сложности сцены;

– работа с поверхностями в математической форме.

Недостатки:

– производительность.

Данный алгоритм не отвечает главному требованию – скорости работы, но при некоторой адаптации скорость работы алгоритма можно повысить.

1.4.4 Алгоритм художника

Данный алгоритм работает аналогично тому, как художник рисует картину – сначала рисуются дальние объекты, затем – ближние. Наиболее распространенная реализация алгоритма — сортировка по глубине, которая заключается в том, что множество граней сортируется по расстоянию от наблюдателя, затем отсортированные грани выводятся на экран в порядке от самой дальней до самой ближней. данный метод хорошо применим для сцен, в которых нет пересекающихся граней.

Преимущества:

– требуется меньше памяти, чем в алгоритме Z-буфера.

Недостатки:

– недостаточно высокая реалистичность изображения;

– сложность реализации при пересечении граней на сцене.

Данный алгоритм не отвечает главному требованию — реалистичности изображения. Также алгоритм художника отрисовывает все грани (в том числе и невидимые), на что тратится большая часть времени.

1.4.5 Вывод

Для удаления невидимых линий выбран алгоритм обратной трассировки лучей. Данный алгоритм позволит добиться максимальной реалистичности и даст возможность смоделировать распространение света в пространстве, учитывая законы геометрической оптики. Данный алгоритм можно модернизировать, добавив в него обработку новых световых явлений.

Также этот алгоритм позволяет строить качественные тени с учетом большого числа источников. Стоит отметить тот факт, что алгоритм трассировки лучей не требователен к памяти, в отличие, например, от алгоритма Z-буфера.

1.5 Анализ и выбор модели освещения

Физические модели материалов стараются аппроксимировать свойства некоторого реального материала. Такие модели учитывают особенности поверхности материала или же поведение частиц материала.

Эмпирические модели материалов устроены иначе, чем физически обоснованные. Данные модели подразумевают некий набор параметров, которые не имеют физической интерпретации, но которые позволяют с помощью подбора получить нужный вид модели.

В данной работе следует делать выбор из эмпирических моделей, а конкретно из модели Ламберта и модели Фонга.

1.5.1 Модель Ламберта

Модель Ламберта моделирует идеальное диффузное освещение, то есть свет при попадании на поверхность рассеивается равномерно во все стороны. При такой модели освещения учитывается только ориентация поверхности (N) и направление источника света (L). Иллюстрация данной модели представлена на рисунке 1.1.

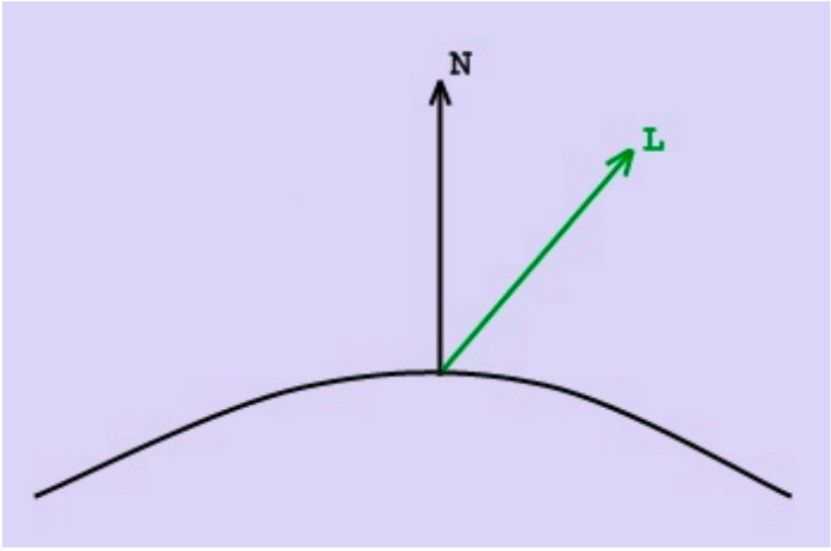

Рисунок 1.1 – Направленность источника света

Эта модель является одной из самых простых моделей освещения и очень часто используется в комбинации с другими моделями. Она может быть очень удобна для анализа свойств других моделей, за счет того, что ее легко выделить из любой модели и анализировать оставшиеся составляющие.

1.5.2 Модель Фонга

Это классическая модель освещения. Модель представляет собой комбинацию диффузной и зеркальной составляющих. Работает модель таким образом, что кроме равномерного освещения на материале могут появляться блики. Местонахождение блика на объекте определяется из закона равенства углов падения и отражения. Чем ближе наблюдатель к углам отражения, тем выше яркость соответствующей точки.

Падающий и отраженный лучи лежат в одной плоскости с нормалью к отражающей поверхности в точке падения (рисунок 1.2). Нормаль делит угол между лучами на две равные части. L – направление источника света, R – направление отраженного луча, V – направление на наблюдателя.

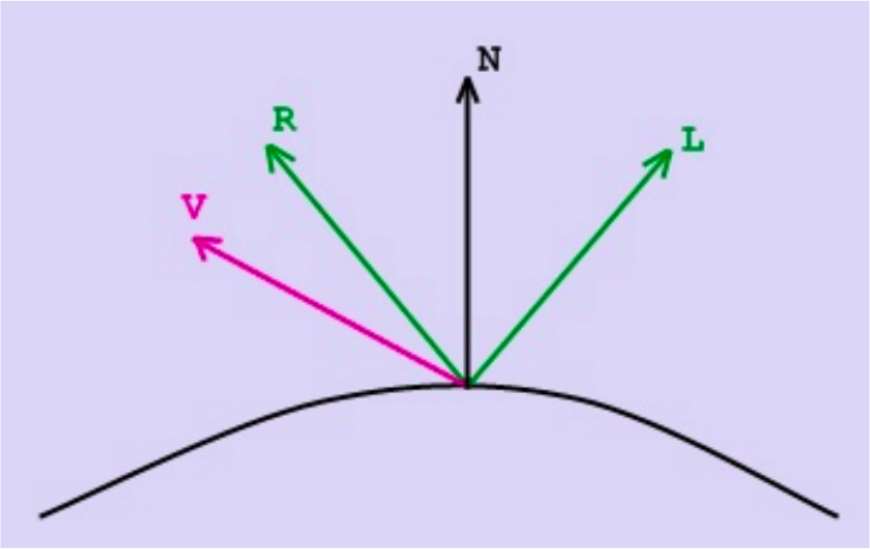

Рисунок 1.2 – Направленность источника света

1.5.3 Вывод

В качестве модели освещения в данной работе была выбрана модель Ламберта из-за своей простоты по сравнению с моделью Фонга. Для расчета данных модели Ламберта необходимо выполнять меньше вычислений, а значит ее реализация потребует меньшего количества времени.

1.6 Вывод

В данном разделе был проведен анализ алгоритмов удаления невидимых линий и модели освещения, которые возможно использовать для решения поставленных задач. В качестве ключевого алгоритма, который также можно оптимизировать, выбран алгоритм обратной трассировки лучей, который будет реализован в рамках данного курсового проекта.

2 Конструкторская часть

2.1 Общий алгоритм решения поставленной задачи

- Задать объекты сцены(мяч, источники света);
- С помощью системы изменения координат наблюдателя задать положение взгляда;
- Рассчитать координаты положения мяча после изменения координат наблюдателя;
- Показать новое положение мяча.

2.2 Алгоритм обратной трассировки лучей

Пусть есть камера и экран, находящийся на расстоянии h от камеры (рисунок 2.1). Для удобства следует разбить экран на квадраты. Далее по очереди необходимо проводить лучи из камеры в центр каждого квадрата (первичные лучи). После этого надо найти пересечение каждого такого луча с объектами сцены и выбрать среди всех пересечений самое близкое к камере. Далее, применив нужную модель освещения, можно получить изображение сцены. Это самый простой метод обратной трассировки лучей. Он позволяет лишь отсечь невидимые грани. Но если надо смоделировать такое явление как отражение, то необходимо из самого близкого пересечения пустить вторичные лучи.

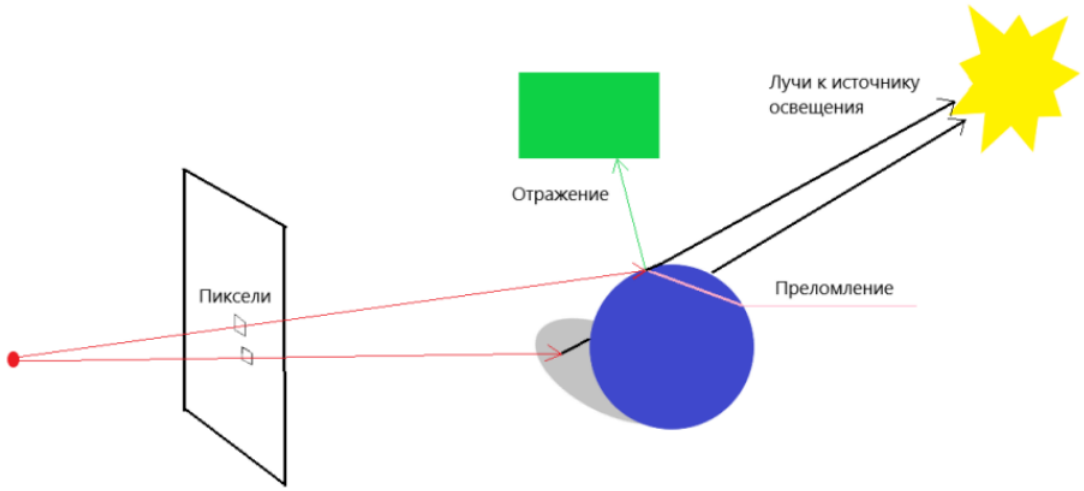

Рисунок 2.1 – Пример работы алгоритма обратной трассировки лучей

Например, если поверхность отражает свет, и она идеально ровная, то необходимо отразить первичный луч от поверхности и пустить по этому направлению вторичный луч. Если же поверхность неровная, то необходимо пустить множество вторичных лучей.

2.3 Способ оптимизации алгоритма обратной трассировки лучей

Для уменьшения времени работы алгоритма при реализации данной программы необходимо испускать лучи из камеры не по всей сцене, а в отдельные ее участки.

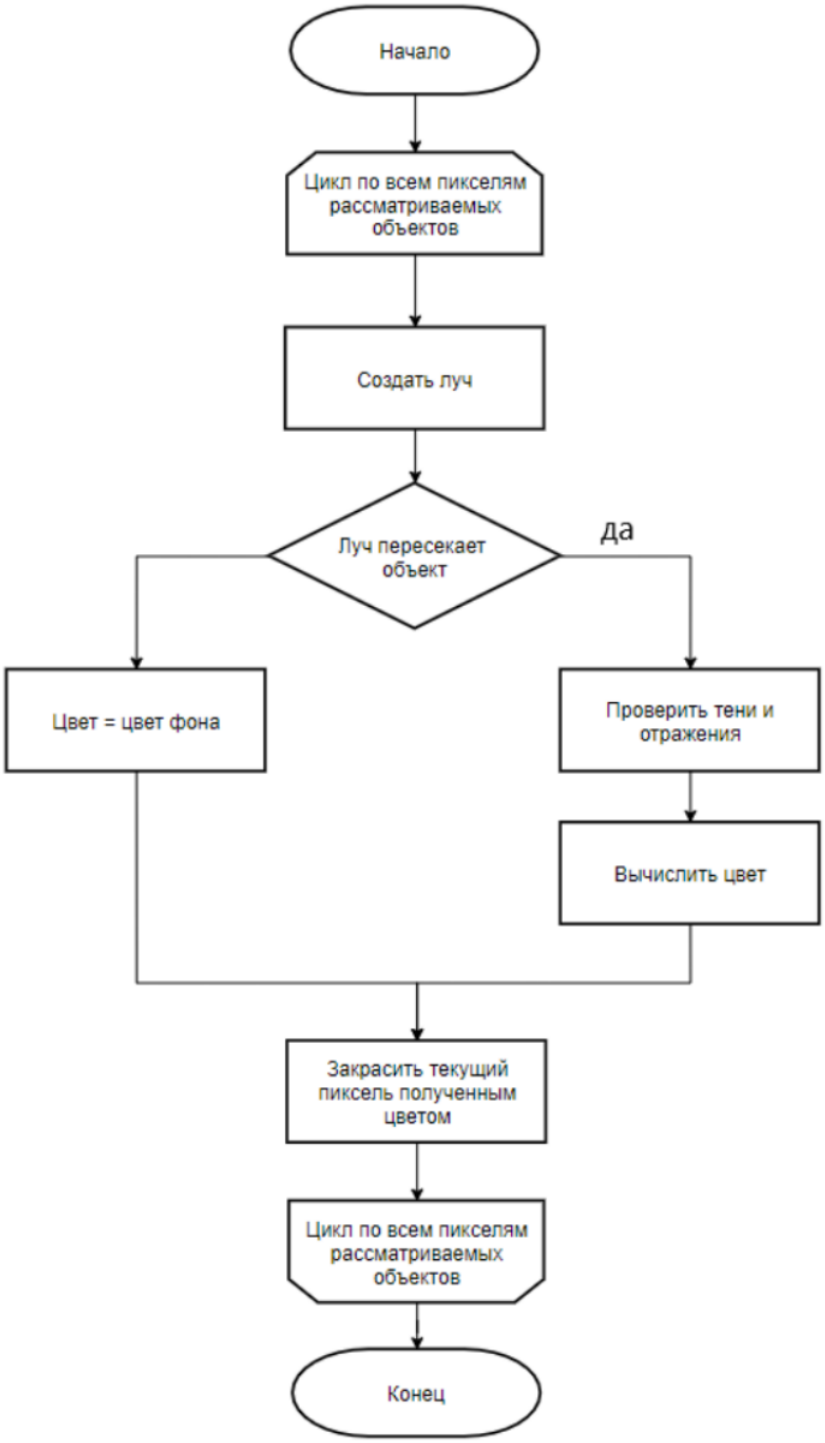

Рисунок 2.2 – Схема алгоритма обратной трассировки лучей

2.4 Модель освещения Ламберта

Данная модель вычисляет цвет поверхности в зависимости от того как на нее светит источник света. Согласно данной модели, освещенность точки равна произведению силы источника света и косинуса угла, под которым он светит на точку.

$$I_d = k_d cos(L, N) i_d,$$

Где:

- I_d – рассеянная составляющая освещенности в точке;
- k_d – свойство материала воспринимать рассеянное освещение;
- i_d – мощность рассеянного освещения;
- L – направление из точки на источник;
- N – вектор нормали.

3 Технологическая часть

В данном разделе обозначаются требования к ПО, обосновывается выбор языка программирования, приводится реализация алгоритмов, схемы которых были разработаны в конструкторской части.

3.1 Требования к программе

Программа должна предоставлять следующие возможности:

– визуальное отображение сцены;

– поворот сцены;

– перемещение взгляда наблюдателя по площади сцены.

3.2 Выбор языка программирования и среды разработки

Существует множество языков, а также сред программирования, многие из которых обладают достаточно высокой эффективностью, удобством и простотой в использовании. Для разработки данной программы был выбран язык C++. Данный выбор обусловлен тем, что в данном языке есть все необходимые инструменты для реализации рассматриваемой задачи, а также опытом применения данного языка программирования.

В качестве среды разработки выбран QT Creator, так как я знаком с данной средой разработки и она позволяет работать с Windows Forms — интерфейсом для создания стандартных Windows приложений.

3.3 Реализация алгоритмов

В листингах приведены соответственно реализации алгоритма обратной трассировки лучей. А именно вычисление поворотов сцены, функция за-

краски, функция проверки пересечения сферы, функция проверки освещения объекта. А также представлена функция работы со слайдерами.

Листинг 3.1 – Работа со слайдерами для перемещения взгляда наблюдателя

```
1  void MainWindow::on_xMoveSlider_valueChanged(int value)
2  {
3      m_sw->m_xView = value;
4      if(!m_freeze) m_sw->repaint();
5  }
6
7  void MainWindow::on_yMoveSlider_valueChanged(int value)
8  {
9      m_sw->m_yView = value;
10     if(!m_freeze) m_sw->repaint();
11 }
12
13 void MainWindow::on_zMoveSlider_valueChanged(int value)
14 {
15     m_sw->m_zView = value;
16     if(!m_freeze) m_sw->repaint();
17 }
18
19 void MainWindow::on_zRotate_valueChanged(int value)
20 {
21     m_sw->m_zRotation = value / 100.0;
22     if(!m_freeze) m_sw->repaint();
23 }
24
25 void MainWindow::on_xRotate_valueChanged(int value)
26 {
27     m_sw->m_xRotation = value / 100.0;
28     if(!m_freeze) m_sw->repaint();
29 }
30
31 void MainWindow::on_yRotate_valueChanged(int value)
32 {
33     m_sw->m_yRotation = value / 100.0;
34     if(!m_freeze) m_sw->repaint();
35 }
```

Листинг 3.2 – Вычисление поворотов сцены

```
point &point::rotateX(float angle)
{
    float newy = y* std::cos(angle) - z * std::sin(angle);
    float newz = y * std::sin(angle) + z * std::cos(angle);
    y = newy;
    z = newz;
    return *this;
}

point &point::rotateY(float angle)
{
    float newx = x* std::cos(angle) - z * std::sin(angle);
    float newz = x * std::sin(angle) + z * std::cos(angle);
    x = newx;
    z = newz;
    return *this;
}

point &point::rotateZ(float angle)
{
    float newx = x* std::cos(angle) - y * std::sin(angle);
    float newy = x * std::sin(angle) + y * std::cos(angle);
    x = newx;
    y = newy;
    return *this;
}
```

Листинг 3.3 – Функция закраски

```
void ScheneWidget::paintEvent(QPaintEvent *)
{
    QPainter painter(this);
    QColor coordLineColor(255, 0, 0, 255);

    QPen apen = QPen(coordLineColor);
    apen.setWidth(1);
    painter.setPen(apen);

    QRect wnd_bounds = painter.window();

    for(int i=-wnd_bounds.width()/2; i < wnd_bounds.width()/2;
        i++) {
```

```
13          for(int j= -wnd_bounds.height()/2; j <
               wnd_bounds.height()/2; j++) {
14            point d = CanvasToViewport(QPoint(i, j), wnd_bounds)
15            .rotateX(m_xRotation)
16            .rotateY(m_yRotation)
17            .rotateZ(m_zRotation);
18            point pov(m_xView,m_yView,m_zView);
19            float t;
20            if (intersectsSphere(pov, d, t) && t>=1) {
21                point inters_point(pov.x + d.x * t, pov.y + d.y *
                     t, pov.z + d.z *t);
22                float ints = getLightning(
23                inters_point,
24                GeomVector(m_sphereCenter, inters_point), false);
25                if ((inters_point.x - m_sphereCenter.x > 0 &&
26                inters_point.z - m_sphereCenter.z < 0) ||
27                (inters_point.x - m_sphereCenter.x < 0 &&
28                inters_point.z - m_sphereCenter.z > 0)) {
29                    painter.setPen(QColor(255 * ints, 0, 0, 255));
30                } else {
31                    painter.setPen(QColor(0, 255 * ints, 0, 255));
32                }
33            } else {
34                point normal;
35                t = findWall(pov, d, normal);
36                point inters_point(pov.x + d.x * t, pov.y + d.y *
                     t,pov.z +  d.z *t);
37
38                float ints = getLightning(
39                inters_point,
40                GeomVector(point(0,0,0), normal), true);
41
42                painter.setPen(QColor(0, 0, 255 * ints, 255));
43            }
44            painter.drawPoint(wnd_bounds.width()/2 + i,
                 wnd_bounds.height()/2 - j);
45          }
46      }
47  }
```

Листинг 3.4 – Функция проверки пересечения сферы

```
1  bool ScheneWidget::intersectsSphere(const point &pov, const point
       &ray, float& t)
2  {
3      GeomVector pov_to_center(m_sphereCenter, pov); // OC
4      GeomVector ray_vector(point(0,0,0), ray); // D
5      auto k1 = ray_vector.multiply(ray_vector);
6      auto k2 = 2*pov_to_center.multiply(ray_vector);
7      auto k3 = pov_to_center.multiply(pov_to_center) -
8      m_sphereRadius * m_sphereRadius;
9      auto discriminant = k2*k2 - 4*k1*k3;
10     if (discriminant < 0) return false;
11     auto t1 = (-k2 + sqrt(discriminant)) / (2*k1);
12     auto t2 = (-k2 - sqrt(discriminant)) / (2*k1);
13     //if (t1 < 1 || t2 < 1) return false;
14     t = std::min(t1, t2);
15     return true;
16 }
```

Листинг 3.5 – Функция проверки освещения объекта

```
1  float ScheneWidget::getLightning(const point &surface_point, const
       GeomVector& normal, bool with_shadows)
2  {
3      float final = 0.2;
4      const float intensity = 0.4;
5
6      {
7          const point light_position(200, -100, 100);
8          point ray(light_position.x - surface_point.x,
9          light_position.y - surface_point.y,
10         light_position.z - surface_point.z); // L
11         float f_dummy = 0;
12         if (with_shadows && !intersectsSphere(surface_point, ray,
               f_dummy) || f_dummy > 1 || f_dummy == 0) {
13             GeomVector light_vector(surface_point, light_position);
14             auto n_dot_l = normal.multiply(light_vector) /
                   normal.getLength();
15             if (n_dot_l > 0) {
16                 final += intensity * n_dot_l /
                       light_vector.getLength();
17             }
18         }
```

3.4 Вывод

В данном разделе были рассмотрены средства реализации ПО и разработаны исходные коды реализации поставленной задачи.

4 Исследовательская часть

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование, следующие:

– Операционная система: Windows 11;

– Память: 16 Гб;

– Процессор: AMD Ryzen 7 6800HS with Radeon Graphics.

4.2 Демонстрация работы программы

В данном разделе приведена демонстрация работы программного обеспечения соответственно.

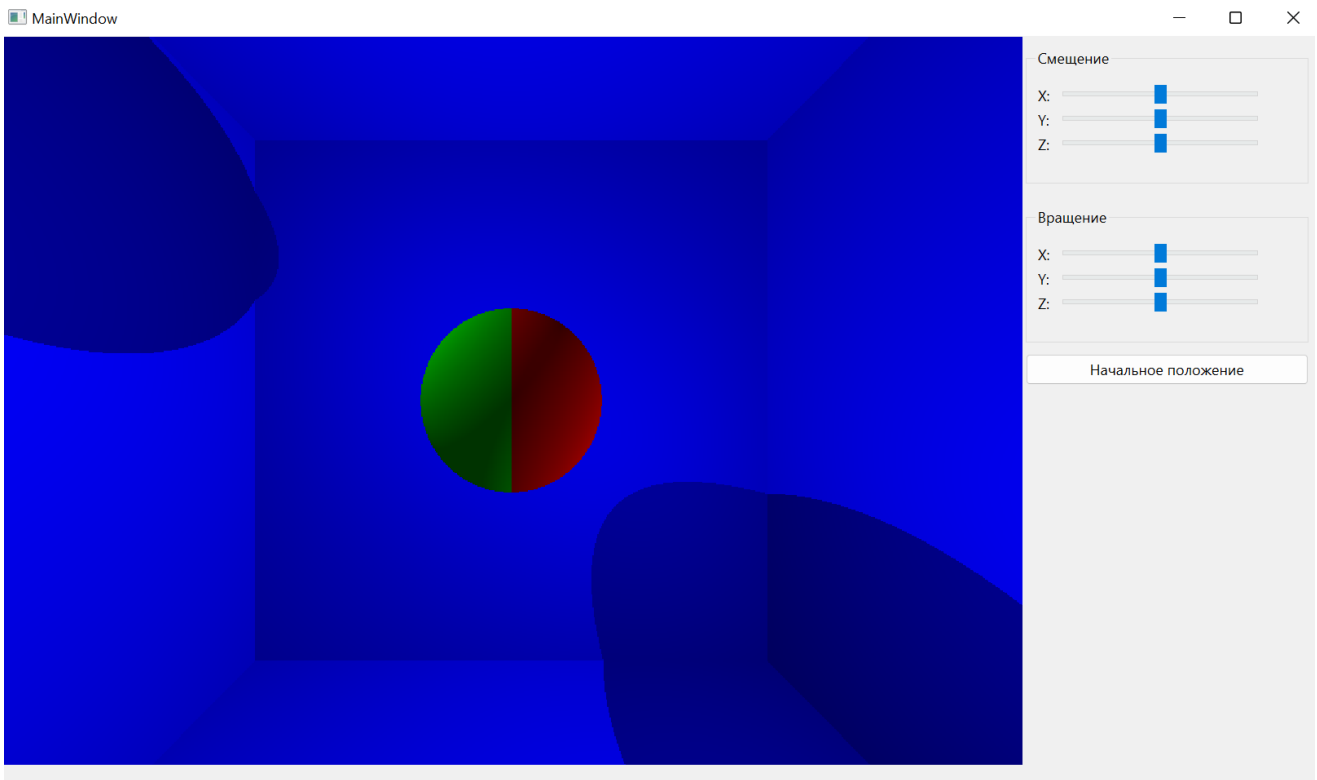

Рисунок 4.1 – Начальный интерфейс программы

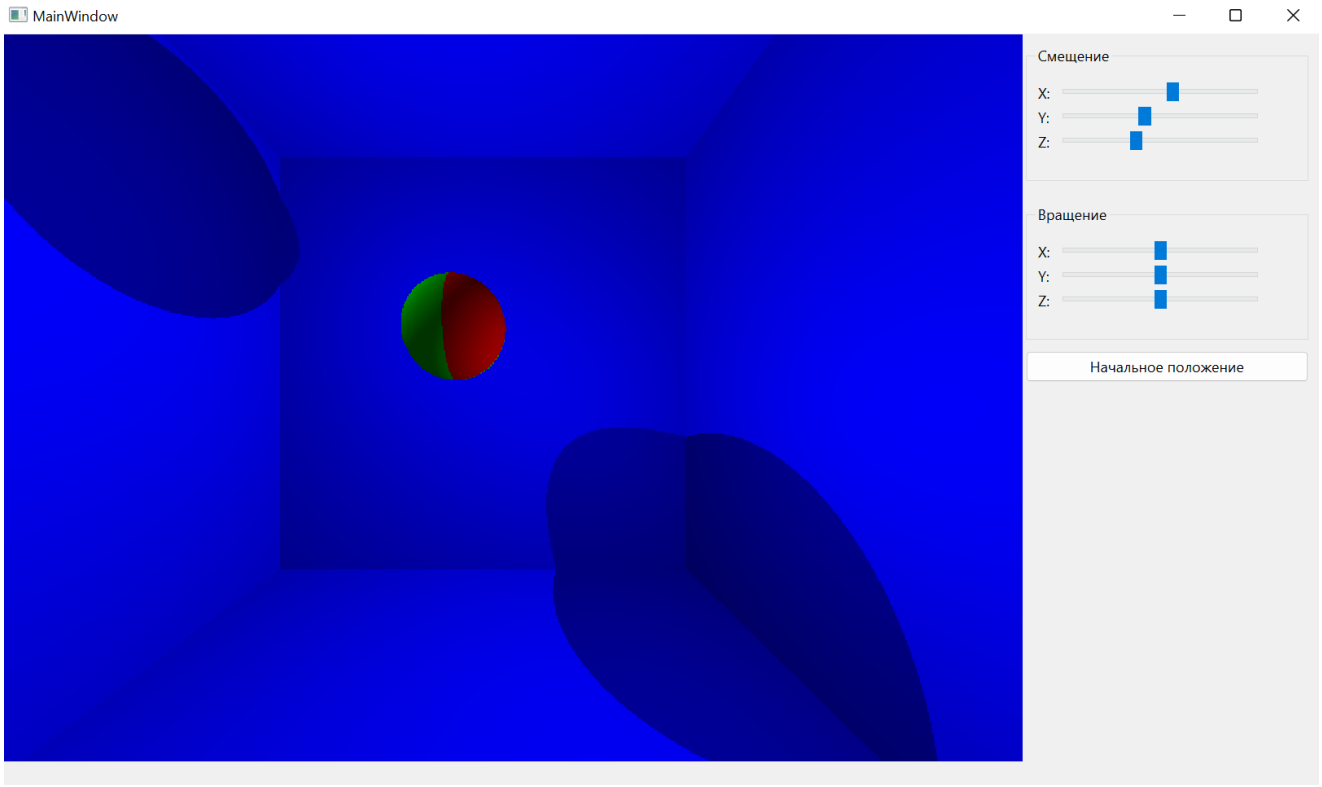

Рисунок 4.2 – Пример работы смещения взгляда наблюдателя

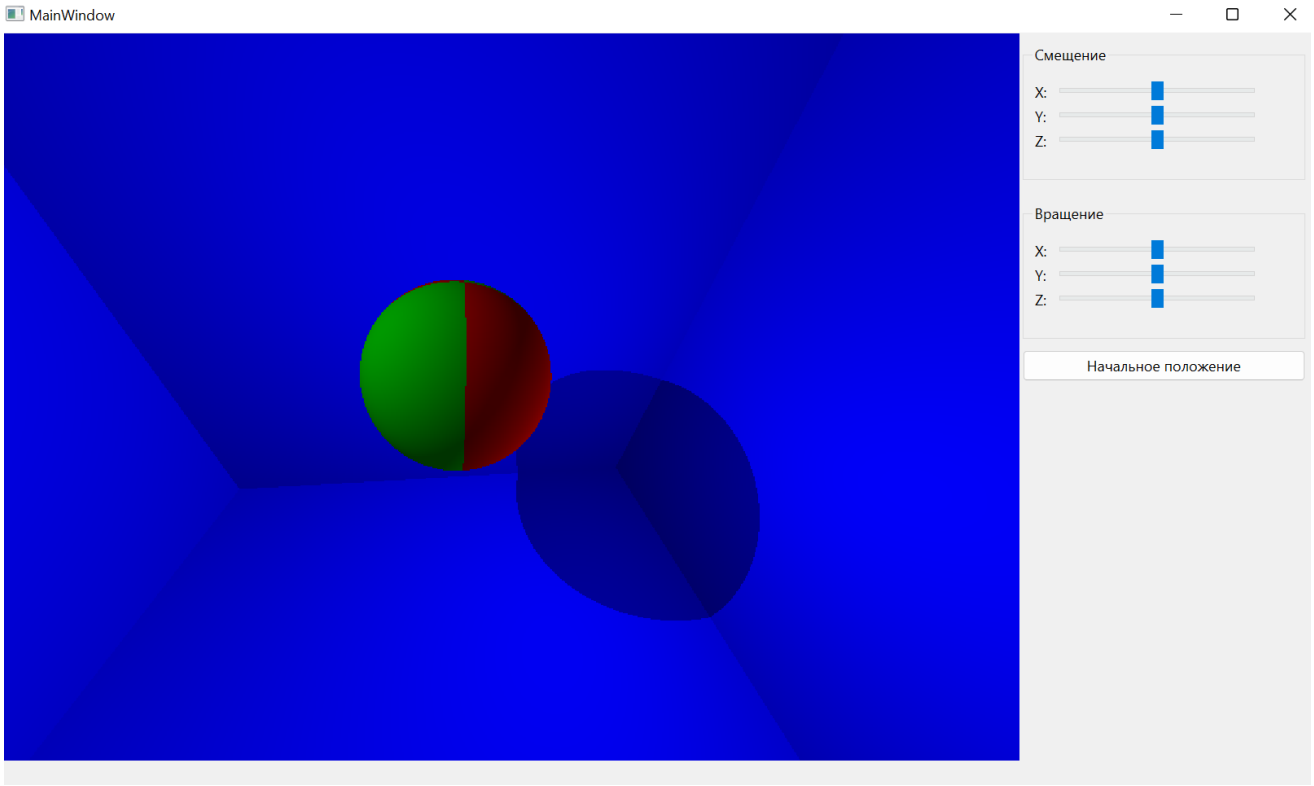

Рисунок 4.3 – Пример вращения сцены

4.3 Замеры времени

Благодаря оптимизации алгоритма обратной трассировки лучей время работы было уменьшено. Так как трассировка данного изображения является

достаточно короткой задачей, используется усреднение времени, за которое проведен массовый эксперимент(10 подходов). Результат представлен на рис. 4.4.

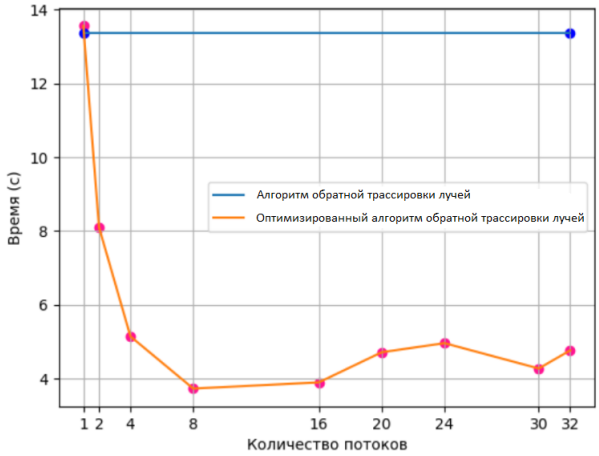

Рисунок 4.4 – Время работы оптимизированной и не оптимизированной реализаций алгоритма обратной трассировки

Оптимизированный алгоритм работает быстрее, так как при данной реализации необходимо испускать лучи не по всей сцене, а в отдельные ее участки. Следовательно не приходится выполнять лишние вычисления.

ЗАКЛЮЧЕНИЕ

Цель работы достигнута: было разработано программное обеспечение, которое строит модель баскетбольного мяча и позволяет рассмотреть ее со всех сторон. В ходе проделанной работы были выполнены все поставленные задачи:

- была описана структура трехмерной сцены;
- были выбраны подходящие к данной задаче алгоритмы трехмерной визуализации;
- были выбраны подходящая среда разработки и язык программирования для решения поставленной задачи;
- были реализованы выбранные алгоритмы визуализации;
- было реализовано программное обеспечение, реализующее выбранные алгоритмы.

Программу можно улучшить внесением следующих изменений:

- распараллеливанием алгоритмов для ускорения работы;
- использованием ресурсов графического процессора.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Алгоритм Z-буфера. [Электронный ресурс]. Режим доступа: http:// compgrap (дата обращения: 09.10.2021).

2. Трассировка лучей из книги Джефа Проузиса [Электронный ресурс]. Режим доступа: https://www.graphicon.ru/oldgr/courses/cg99/ notes/lect12/p (дата обращения: 28.10.2021).

3. Алгоритм Робертса. [Электронный ресурс]. Режим доступа: http:// compgrap (дата обращения: 09.10.2021).

4. Алгоритм, использующие список приоритетов (алгоритм художника). [Электронный ресурс]. Режим доступа: http://compgraph.tpu.ru/ Oglavlenie.h (дата обращения: 09.10.2021).

5. Алгоритм Варнока. [Электронный ресурс]. Режим доступа: http:// compgrapl (дата обращения: 09.10.2021).

6. Модели освещения. [Электронный ресурс]. Режим доступа: https:// devburn.r (дата обращения: 09.10.2021)

7. Windows [Электронный ресурс]. Режим доступа: https://www. microsoft.com/ ru/windows (дата обращения: 30.09.2021).

8. Процессор Intel® Core™ i5-1135G7 [Электронный ресурс]. Режим доступа: https://www. intel.ru/content/www/ru/ru/products/sku/208658/ intel-core-i51135g7-processor-8m-cache-up-to-4-20-ghz/ specifications.html (дата обращения: 04.09.2021).