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»_____

РАСЧЁТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОМУ ПРОЕКТУ
НА ТЕМУ:

«Моделирование конструктора лего»

Студент _____ИУ7-55Б_____ _____ __Платонова О. С.__
(Группа) (подпись, дата) (фамилия, и.о.)

Руководитель курсового проекта _____ ___Шикуть А. В._____
(подпись, дата) (фамилия, и.о.)

2020 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)

УТВЕРЖДАЮ
Заведующий кафедрой __ИУ7____
(Индекс)
_____ И.В._Рудаков_
(И.О.Фамилия)
« _____ » _____ 20 ____ г.

З А Д А Н И Е
на выполнение курсового проекта

по дисциплине _____Компьютерная графика_____

Студент группы ____ИУ7-55Б_____

_____Платонова Ольга Сергеевна_____
(Фамилия, имя, отчество)

Тема курсового проекта _____”Моделирование конструктора лего”_____
_____
_____

Направленность КП (учебный, исследовательский, практический, производственный, др.)
_____учебный_____
Источник тематики (кафедра, предприятие, НИР) ___кафедра_____

График выполнения проекта: 25% к _4_ нед., 50% к _7_ нед., 75% к _11 нед., 100% к _14 нед.

Задание

Разработать программное обеспечение для создания реалистичной модели конструктора. Предоставить пользователю возможность работы со сценой: добавление, изменение, удаление деталей. Доступные пользователю детали указать в меню программы. Детали считать выпуклыми. Предоставить возможность закраски детали в выбранный цвет. Предусмотреть возможность поворота на заданный угол вокруг заданной оси, переноса на заданные координаты отдельно взятой детали и сцены целиком. При визуализации источник освещения расположить на бесконечности. Окно программы должно иметь фиксированный размер.

Оформление курсового проекта:

Расчетно-пояснительная записка на 25-30 листах формата А4.
Расчетно-пояснительная записка должна содержать постановку введение, аналитическую часть, конструкторскую часть, технологическую часть, экспериментально-исследовательский раздел, заключение, список литературы, приложения.
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 «__ » _____ 20__г.

Руководитель курсового проекта _____ ____А. В. Шикуть____
(Подпись, дата) (И.О.Фамилия)

Студент _____ ___О. С. Платонова___
(Подпись, дата) (И.О.Фамилия)

Оглавление

Введение

Компьютерная графика в настоящее время уже вполне сформировалась как наука. Существует аппаратное и программное обеспечение для получения различных изображений – от простых чертежей до реалистичных объектов трехмерного пространства. Компьютерная графика затрагивает практически все дисциплины: от биологии и медицины, до бизнеса и индустрии развлечений.

В связи с этим возникает задача разработки алгоритмов, которые смогут быстро, с минимальными затратами ресурсов построить качественное трехмерное реалистичное изображение. Построение такого изображения включает в себя как физические, так и психологические процессы. Повышение реалистичности изображения может быть достигнуто с учетом оптических явлений преломления, отражения и рассеивания света, путем воспроизведения теней, свойств и текстуры поверхностей.

На данный момент разработано множество алгоритмов, решающих задачу построения реалистичного изображения трехмерного пространства.

Целью данной курсовой работы является разработка приложения для моделирования конструктора из заданного набора деталей.

Для реализации указанной цели необходимо решить следующие задачи:

- разработать структуру программного продукта, реализующего построение конструктора на трехмерной сцене;
- проанализировать существующие алгоритмы, позволяющие выполнять построение трехмерной сцены;
- реализовать выбранные алгоритмы, выполняющие построения на сцене;
- разработать программное обеспечение, соответствующее заданным критериям.

1. Аналитическая часть

В данном разделе рассмотрим задачи и методы их решения визуализации сцены, которые будет выполнять разработанное программное обеспечение.

1.1. Формализация объектов сцены

Для описания трёхмерных геометрических объектов существует три модели: каркасная, поверхностная и объемная. Для реализации поставленной задачи воспользуемся поверхностной моделью, которая позволяет строить более реалистичное изображение. Реализация поверхностной модели возможно с помощью полигональной сетки и параметрической функции.

В случае работы с полигональной сеткой выделяют три типа представления:

1. Вершинное представление. Вершины хранят указатели на соседние вершины.
2. Список граней. Объект хранится как множество граней и вершин.
3. Таблица углов. Вершины хранятся в предопределённой таблице, обход которой задает полигоны неявно.

В поставленной задаче наиболее уместным способ представления трехмерных объектов является представление в виде списка граней. Это представление позволяет эффективно манипулировать данными.

Объекты сцены:

1. Деталь. Трехмерный объект заданной формы. Информация об объекте хранится в файле. Объект видимый.
2. Источники света. Испускает параллельные лучи, от угла падения которых зависит интенсивность цвета и тени деталей. Описывается точкой и углом наклона. Объект невидимый.

1.2. Постановка задачи

Требуется реализовать программное обеспечение, позволяющее выполнять следующие операции:

- добавлять на сцену детали из списка заданных;
- вращать, масштабировать и перемещать выбранную деталь в произвольном направлении;
- добавлять на сцену источник освещения и перемещать на заданные координаты;
- вращать сцену в произвольном направлении.

При добавлении нового элемента на сцену, может возникнуть ситуация, когда элементы, установленные ранее, окажутся для пользователя невидимыми полностью или частично. Также, во время вращения сцены, набор невидимых элементов будет изменяться. Для решения поставленной задачи, требуется алгоритм, позволяющий быстро удалять невидимые линии и поверхности.

1.3. Анализ алгоритмов удаления невидимых линий и поверхностей

Задача удаления невидимых линий и поверхностей является одной из наиболее сложных в машинной графике, что привело к большому числу различных способов ее решения.

Выбор подходящего решения следует основывать на особенностях указанной задачи: скорость работы. Поскольку пользователь выполняет вращение сцены в произвольном направлении с произвольной скоростью, от алгоритма требуется быстрота работы. Также известно, что на сцене могут присутствовать произвольные тела вращения. При этом следует учитывать, что

скорость и детальность реализации обратно пропорциональны, поэтому построенная модель не сможет достигнуть хорошей детальности.

1.3.1. Алгоритм Робертса

Алгоритм работает в объектном пространстве и выполняет удаление сначала из каждого тела те ребра или грани, которые экранируются самим телом. Затем каждое из видимых ребер сравнивается с каждым из оставшихся тел для определения того, какая его часть экранируются этими телами.

Вычислительная трудоемкость алгоритма растет как квадрат числа объектов. Поскольку число объектов на сцене не ограничено, и возможно построение сложных фигур с большим числом элементов, алгоритм Робертса не сможет поддержать требуемую быстроту вычислений. Однако, стоит отметить простоту и точность методов, используемых в указанном алгоритме.

1.3.2. Алгоритм Варнока

В алгоритме осуществляется попытка извлечь преимущество из факта однородности больших областей изображения. В пространстве изображений рассматривается окно и решается вопрос: пусто оно, или его содержимое достаточно просто. Если это не так, то окно разбивается на фрагменты до тех пор, пока ответ на вопрос не станет утвердительным.

Было предложено множество модификаций алгоритма, связанных с вариантами разбиения окна. Несмотря на то, что алгоритм достаточно прост и понятен, рекурсивное разбиение окна в некоторых случаях будет работать недостаточно быстро для динамической работы со сценой. Множественное разбиение окна может возникнуть при обработке деталей, которые являются сложной конструкцией, содержащей цилиндры.

1.3.3. Алгоритм, использующий z-буфер

Рассматриваемый алгоритм является одним из простейших алгоритмов, работающих в пространстве изображения. Работа алгоритма основывается на идеи создания отдельного буфера глубины, используемого для запоминания глубины каждого видимого пиксела. В процессе работы значение z каждого нового пиксела, претендующего на занесение в буфер, сравнивают с глубиной того пиксела, который уже занесен в z-буфер. Несмотря на многочисленные сравнения пикселов по глубине, скорость алгоритма высока из-за отсутствия предварительной сортировки. Следовательно, сцены могут быть любой сложности.

Основной недостаток – большой объем требуемой памяти, однако на сегодняшний день использование большого объема оперативной памяти не является существенным недостатком предложенного алгоритма.

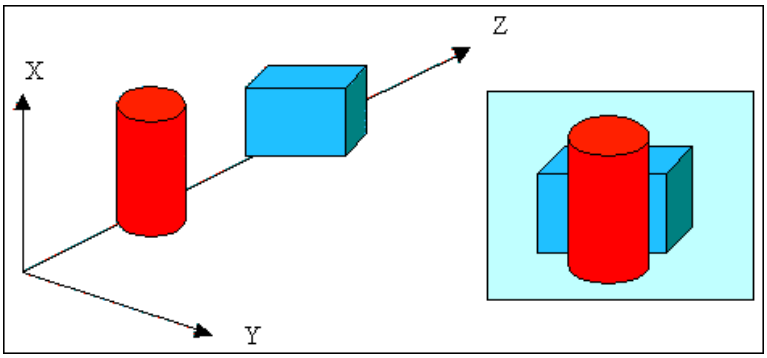


Рисунок 1.3.3.1. Пример работы алгоритма z-буфера.

1.3.4. Алгоритм обратной трассировки лучей

Идея алгоритма заключается в том, что происходит наблюдение за траекторией каждого луча света, испускаемого от наблюдателя. В случае, если луч пересекает объект сцены, то определяются все возможные точки пересечения луча и объекта. Наиболее важным и долгим этапом в работе алгоритма является процедура определения пересечений.

Поскольку рассмотренный алгоритм не использует того обстоятельства, что некоторые грани многогранника являются нелицевыми и их можно сразу удалить, он не является оптимальным решением указанной задачи. Алгоритм предназначен для решений более общих задач, что в данном случае приведет к излишним вычислениям, и, как следствие, к замедлению времени работы.

1.3.5. Используемый алгоритм удаления невидимых граней

При реализации программного продукта используется алгоритм z-буфера, позволяющий довольно легко и быстро осуществить удаление невидимых граней и плоскостей, независимо от количества деталей на сцене. В качестве предварительного шага удаляются нелицевые грани, что снижает количество сравнений и увеличивает скорость работы. Также для увеличения скорости, вновь добавленную деталь следует заносить не в конец списка элементов на сцене, а в соответствии с ее расположением относительно z-координаты. Таким образом, оценка вычислительной трудоемкости алгоритма не более, чем линейна.

1.4. Анализ алгоритмов закраски

Поскольку модель конструктора предполагает, что основными пользователями являются дети – цвета деталей должны быть яркими и насыщенными. Рассмотрим различные алгоритмы закраски, позволяющие быстро и натурально выполнять закрашивание детали.

Метод Гуро основан на билинейной интерполяции, с помощью которой вычисляется интенсивность каждого пиксела на сканирующей строке. Такие недостатки метода, как эффект полос Маха и одинаковая интенсивность на складчатой поверхности, будут незначительны в указанной задаче, потому что на сцене отсутствуют массивные тела вращения и расположены лишь выпуклые

фигуры. Однако, в ключе поставленной задачи возникает и другая проблема – сглаживание изображения.

Закраска Фонга позволяет разрешить проблемы метода Гуро, путем вычисления билинейной интерполяцией вектора нормали, что приводит к большим вычислительным затратам. Также указанный метод позволяет получить более реалистичное изображение, в том числе правдоподобнее выглядят зеркальные блики.

Работа метода *простой модели освещения* заключается в закрашивании всей грани одним уровнем интенсивности, рассчитывающимся по закону Ламберта. В результате использования этого метода поверхность предметов на сцене выглядит матовой. Выбор простой модели позволяет экономить ресурсы: сокращает время работы алгоритма (не требуются вычисления нормалей и перебор сканирующих строк), уменьшает объем требуемой памяти (что важно при реализации z-буфера, требующего большое количество памяти), а также не выполняет сглаживание. Однако указанный метод плохо учитывает отражения и при отрисовки тел вращения возникают проблемы.

1.5. Модель освещения

Существуют локальная и глобальная модели освещения, которые различны в учете факторов при расчете интенсивности отраженного к наблюдателю света в каждой точке. В случае локальной модели во внимание принимается только свет, падающий от источника, и ориентация поверхности, при реализации глобальной модели – учитывается также свет, отраженный от других объектов сцены.

Глобальная модель освещения является более точной, так как передает реальное распространение света. Но поскольку в указанной задаче ведется работа с матовыми поверхностями, точность локальной модели достаточна для визуализации рассматриваемых объектов. Так же, в отличие от глобальной

модели, не требуется трассировка лучей, что позволяет ускорять работу программы.

1.6. Вывод

В данном разделе был произведен сравнительный анализ алгоритмов удаления невидимых поверхностей и ребер, алгоритмов закраски и построения теней. В результате анализа было принято решение о реализации алгоритма Z-буфера (в связи с его простотой и скоростью, относительно других алгоритмов, и возможностью на сегодняшний день предоставлять необходимый объем оперативной памяти) и метода закраски Гуро (благодаря которому криволинейные поверхности будут более гладкими, а также хорошим сочетанием с алгоритмом z буфера). Было принято решение о работе с локальной моделью освещенности в целях ускорения работы программы.

2. Конструкторская часть

Разработаем алгоритмы, которые будут использованы в программном обеспечении, с учетом описанных выше рассуждений.

2.1. Требования к программному обеспечению

Прежде всего, программное обеспечение должно быть понятно пользователю: надписи меню должны быть на русском языке, должно быть доступно два режима: моделирования и обзора. Также программа должна предоставлять следующие возможности:

- добавление новой детали на сцену;
    - вращение детали;
    - масштабирование детали;
    - перемещение детали;
- добавление точечных источников света;
    - перемещение источников;
    - изменение интенсивности;
- вращение исходной сцены в произвольном направлении.

2.2. Преобразования деталей

Ранее мы указывали, что вновь помещаемая на сцену деталь должна быть доступна пользователю следующим образом:

- поворот;
- масштабирование;
- перенос.

Для реализации перечисленных операций воспользуемся формулами преобразований, изученными в курсе компьютерной графики.

Поворот

Формулы поворота точки (x; y; z) на угол a.

Вокруг оси Ox

$$\begin{cases} x_1 = x \\ y_1 = y\cos a + z\sin a \\ z_1 = y\sin a + z\cos a \end{cases} \quad (1)$$

Вокруг оси Oy

$$\begin{cases} x_1 = x\cos a + z\sin a \\ y_1 = y \\ z_1 = -x\sin a + z\cos a \end{cases} \quad (2)$$

Вокруг оси Oz

$$\begin{cases} x_1 = x\cos a - y\sin a \\ y_1 = -x\sin a + y\cos a \\ z_1 = z \end{cases} \quad (3)$$

Масштабирование

Формулы масштабирования

$$\begin{cases} x_1 = x * kx \\ y_1 = y * ky \\ z_1 = z * kz \end{cases} \quad (4)$$

где *kx*, *ky*, *kz* – коэффициенты масштабирования.

Перенос

Формулы переноса точки (x; y; z) в точку $(x_1;\ y_1;\ z_1)$

$$\begin{cases} x_1 = x + dx \\ y_1 = y + dy \\ z_1 = z + dz \end{cases} \quad (5)$$

где dx, dy, dz – смещение соответственно по оси Ox, Oy, Oz.

2.3. Преобразования сцены

Аналогично отдельно взятой детали, пользовать должен иметь возможность работы со сценой, а именно - вращать сцену.

Формулы поворота для детали верны и в случае поворота сцены, так как под вращением сцены подразумевается вращение всех элементов сцены.

2.4. Разработка алгоритма, использующего z-буфер

1. Удалить нелицевые грани.
2. Заполнить z-буфер фоновым значением интенсивности.
3. Проинициализировать z-буфер минимальным значением z.
4. Преобразовать деталь в растровую форму в порядке добавления на сцену.
5. Вычислить глубину каждого пиксела многоугольника.
6. Сравнить глубину z(x, y) с соответствующим значением Z_b(x, y) в z-буфере.
    a. Если z(x, y) > Z_b(x, y), то записать атрибут этого многоугольника (интенсивность, цвет и т. п.) в буфер кадра и заменить Z_b(x, y) на z(x, y).
7. Отобразить результат.

Вычисление глубины пиксела

Для вычисления глубины каждого пиксела на сканирующей строке воспользуемся следующими формулами:

$$ax + by + cz + d = 0 \qquad (5)$$

$$z = \frac{-(ax + by + d)}{c} \neq 0$$

Поскольку y = const (сканирующая строка), x_1 = x + Δx (следующий пиксел сканирующей строки), то глубину пиксела можно рассчитать по формуле:

$$z_1 - z = \frac{-(ax+d)+(ax+d)}{c} = \frac{a(x-x_1)}{c}$$

$$z_1 = z - \frac{a}{c}\Delta x \qquad (6)$$

Формула *(5)* – уравнение плоскости.

2.5. Разработка алгоритма закраски Гуро

1. Вычислить нормаль к каждой грани.
2. Вычислить интенсивность в каждой вершине.
3. Для каждой грани:
    a. Вычислить минимальное и максимальное значение *Y* грани.
    b. Для каждой сканирующей строки:
        i. Вычислить минимальное и максимальное значение *X* пересечения строки и грани;
        ii. Вычислить значение интенсивность в точках *Xmin*, *Xmax*;
        iii. Для каждого пикселя в сканирующей строке:
            1. Вычислить интенсивность пикселя;
4. Вывести буфер кадра на экран.

2.6. Разработка общего алгоритма визуализации сцены

Рассмотрим общий алгоритм визуализации объектов сцены.

1. Установка первоначальных объектов сцены.
2. Установка источника света, положения наблюдателя.
3. Добавление детали на сцену:
    a. выбор параметров детали (форма);
    b. вращение детали, перемещение по сцене;
    c. добавление детали в список зарегистрированных на сцене;
4. Для каждого полигона высчитать нормаль и интенсивность цвета;

5. Применить алгоритм z-буфера:
    a. получить изображение сцены;
    b. получить буфер глубины;
6. Применить алгоритм z-буфера:
    a. получить буфер глубины для источника света;
7. Получить список невидимых элементов;
8. Отобразить изображение;
9. Вращение сцены:
    a. применить алгоритм вращения ко всем элементам сцены;
    b. вернуться к п. 2;
10. Конец.

Каждая деталь описывается структурой Detail, содержащей поля:

- имя файла, из которого происходит загрузка;
- координаты переноса;
- координаты масштабирования;
- цвет детали.

Описание сцены – класс Scene:

- вектор деталей на сцене;
- вектор точечных источников на сцене;
- объект камера.

2.7. Вывод

В данном разделе были разработаны алгоритмы удаления невидимых граней и поверхностей, закраски, работы с деталью и сценой; и был представлен общий алгоритм работы программного продукта.

3. Технологическая часть

В данном разделе будут рассмотрены средства реализации, интерфейс программы, а также представлены листинги основных этапов реализации.

3.1. Средства реализации.

Для реализации проекта был выбран язык C++ – компилируемый статически типизированный язык программирования общего назначения. Поддерживает такие парадигмы программирования, как процедурное программирование, объектно-ориентированное программирование, обобщённое программирование. Язык имеет богатую стандартную библиотеку, которая включает в себя распространённые контейнеры и алгоритмы, ввод-вывод, регулярные выражения, поддержку многопоточности и другие возможности.

C++ сочетает свойства как высокоуровневых, так и низкоуровневых языков. В сравнении с его предшественником — языком C — наибольшее внимание уделено поддержке объектно-ориентированного и обобщённого программирования. Также выбор в пользу этого языка был связан с опытом написания на нем и удобством представления детали экземпляром класса.

Средой разработки была выбрана «Qt Creator». Среда включает в себя графический интерфейс отладчика и визуальные средства разработки интерфейса как с использованием QtWidgets, так и QML.

В Qt Creator реализовано автодополнение, в том числе ключевых слов, введённых в стандарте C++11, подсветка кода. Также, начиная с версии 2.4, есть возможность задания стиля выравнивания, отступов и постановки скобок. Все это делает среду удобной и практичной для решения поставленной задачи.

3.2. Реализация основных этапов визуализации.

В листинге 3.2.1 представлен этап вычисления глубины каждой точки полигона и ее интенсивности.

Листинг 3.2.1.

```
void Drawer::triangleProcessing(Vector3i& t0, Vector3i& t1, Vector3i& t2,
                                const QColor& color, float& i0, float& i1, float& i2)
{
    if (t0.y == t1.y && t0.y == t2.y) {
        return;
    }

    if (t0.y > t1.y) {
        swap(t0, t1);
        swap(i0, i1);
    }

    if (t0.y > t2.y) {
        swap(t0, t2);
        swap(i0, i2);
    }

    if (t1.y > t2.y) {
        swap(t1, t2);
        swap(i1, i2);
    }

    int totalHeight = t2.y - t0.y;

    for (int i = 0; i < totalHeight; i++) {
        bool secondHalf = i > t1.y - t0.y || t1.y == t0.y;
        int segmentHeight = secondHalf ? t2.y - t1.y : t1.y - t0.y;

        float alpha = (float)i / totalHeight;
        float betta = (float)(i - (secondHalf ? t1.y - t0.y : 0)) / segmentHeight;

        Vector3i A = t0 + Vector3f(t2 - t0) * alpha;
        Vector3i B = secondHalf ? t1 + Vector3f(t2 - t1) * betta : t0 + Vector3f(t1 - t0) * betta;

        float iA = i0 + (i2 - i0) * alpha;
        float iB = secondHalf ? i1 + (i2 - i1) * betta : i0 + (i1 - i0) * betta;

        if (A.x > B.x) {
            swap(A, B);
            swap(iA, iB);
        }
```

```
      A.x = max(A.x, 0);
      B.x = min(B.x, w);

      for (int j = A.x; j <= B.x; j++) {
         float phi = B.x == A.x ? 1. : (float)(j - A.x) / (float)(B.x - A.x);

         Vector3i P = Vector3f(A) + Vector3f(B - A) * phi;
         float iP = iA + (iB - iA) * phi;

         if (P.x >= w || P.y >= h || P.x < 0 || P.y < 0) {
            continue;
         }

         if (zBuffer.getDepth(P.x, P.y) < P.z) {
            zBuffer.setDepth(P.x, P.y, P.z);
            colorCache[P.x][P.y] = QColor(iColor(color.rgba(), iP));
         }
      }
   }
}
```

В листинге 3.2.2 представлен этап преобразования мировых координат вершин модели в репер камеры, а затем перспективного искажения.

Листинг 3.2.2.

```
void Drawer::objectProcessing(Figure& model, Vector3f& camPos, Vector3f& camDir, Vector3f& camUp)
{
   bool skip = false;

   float camZInc = fabs(camPos.z) + 1;
   float camZDec = fabs(camPos.z) - 1;

   Vector3f center = model.getCenter();
   int faces = model.countFaces();
   QColor color = model.getColor();

   Matrix viewPort = Camera::viewport(w/8, h/8, w*3/4, h*3/4);
   Matrix projection = Matrix::identity(4);
   Matrix modelView = Camera::lookAt(camPos, camDir, camUp);

   projection[3][2]  = - 1.f / (camPos - camDir).norm();

   Matrix mvp = viewPort * projection * modelView;

   for (int i = 0; i < faces; i++) {
      skip = false;
      std::vector<int> face = model.face(i);
```

```
        Vector3i screenCoords[3];
        float intensity[3] = { BG_LIGHT, BG_LIGHT, BG_LIGHT };

        for (int j = 0; j < 3; j++) {
            Vector3f v = center + model.vert(face[j]);

            if (v.z > camZDec && v.z > camZInc) {
                skip = true;
                break;
            }

            screenCoords[j] = Vector3f(mvp * Matrix(v));
            intensity[j] = lightProcessing(v, model.norm(i, j));
        }

        if (skip || !isVisible(screenCoords[0]) ||
                !isVisible(screenCoords[1]) ||
                !isVisible(screenCoords[2]))
            continue;

        triangleProcessing(screenCoords[0], screenCoords[1], screenCoords[2],
                color, intensity[0], intensity[1], intensity[2]);
    }
}
```

В листинге 3.2.3 представлен этап вычисления интенсивности вершины от источников света на сцене.

Листинг 3.2.3.

```
float Drawer::lightProcessing(const Vector3f& vert, const Vector3f& norm)
{
    float wholeIntensity = 0;
    float intensity = 0;

    int lights = scene.countLightSource();

    for (int i = 0; i < lights; i++) {
        intensity = 0;
        LightSourcePoint lsp = scene.getLightSource(i);

        Vector3f lightDir = vert - lsp.getPosition();

        intensity += lightDir * norm / pow(lightDir.norm(), 2.0);
        intensity *= lsp.getIntensity() * LIGHT_REFLECT;

        intensity = fmax(0.0, intensity);
        intensity = fmin(1.0, intensity);
```

```
        intensity = BG_LIGHT + intensity * (1 - BG_LIGHT);

        wholeIntensity += intensity;
    }

    if (wholeIntensity == 0) {
        wholeIntensity = BG_LIGHT;
    }
    else {
        wholeIntensity /= lights;
    }

    return wholeIntensity;
}
```

3.3. Интерфейс программы

На рисунке 3.3.1 представлен интерфейс основного окна программы.

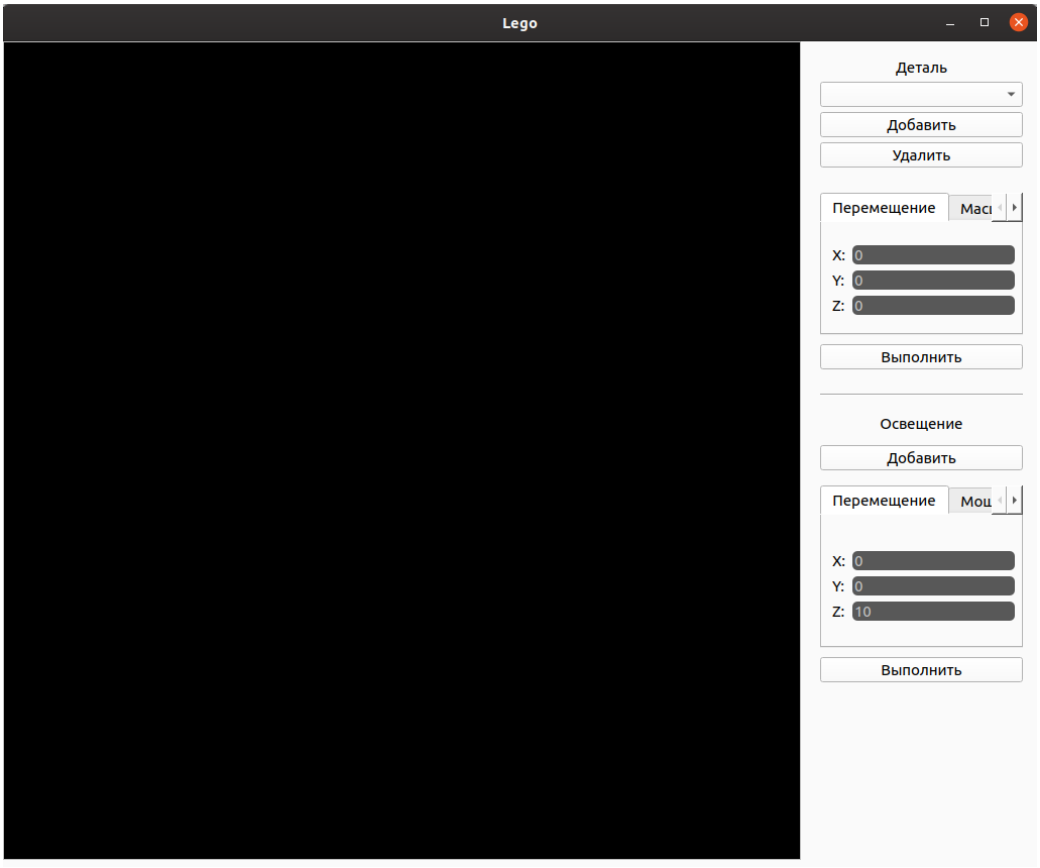


Рис. 3.3.1. Интерфейс основного окна.

Вывод

В данном разделе были рассмотрены средства реализации, представлены листинги основных этапов проектирования, а также представлен интерфейс программы.

4. Исследовательская часть

В данном разделе будут рассмотрены примеры работы программы, произведено тестирование, выполнены эксперименты по замеру времени, а также выполнен сравнительный анализ реализуемых алгоритмов.

4.1. Примеры работы приложения

На рисунках 4.1.1 – 4.1.2 представлены результаты работы приложения после добавления деталей и точечного источника света на сцену с последующими преобразованиями.

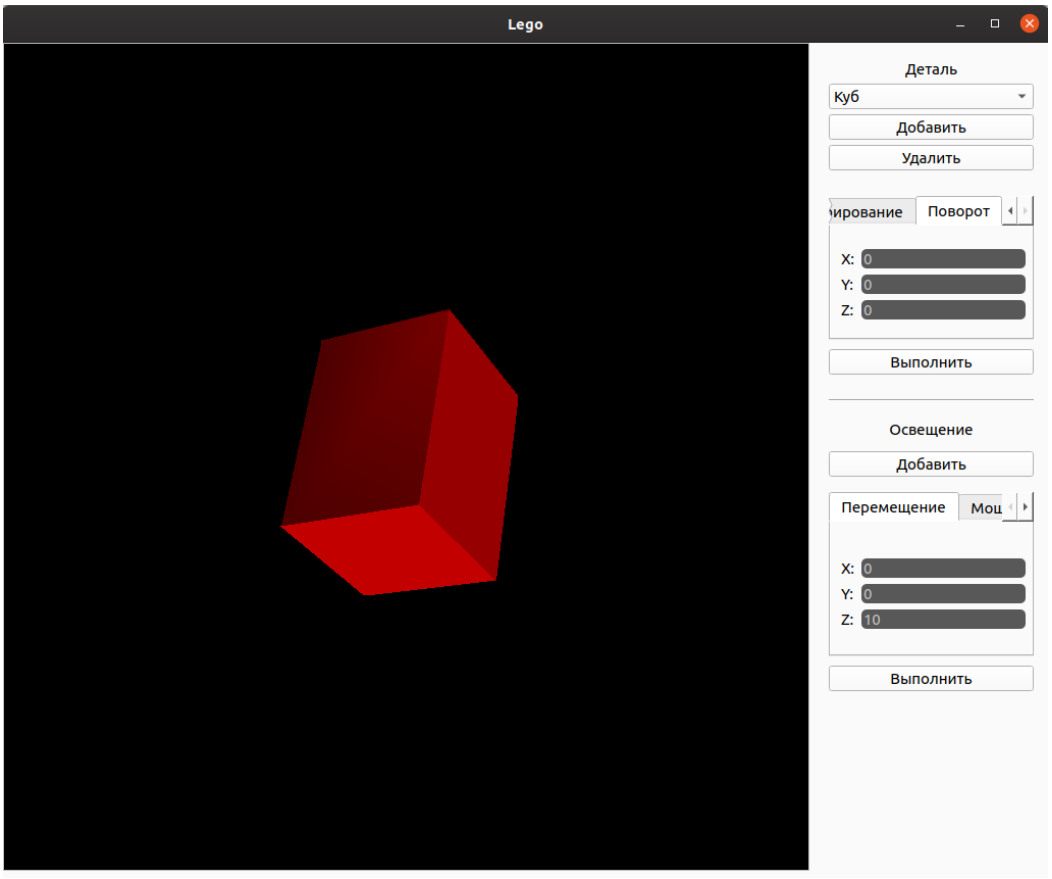


Рисунок 4.1.1. Пример работы приложения. Куб.

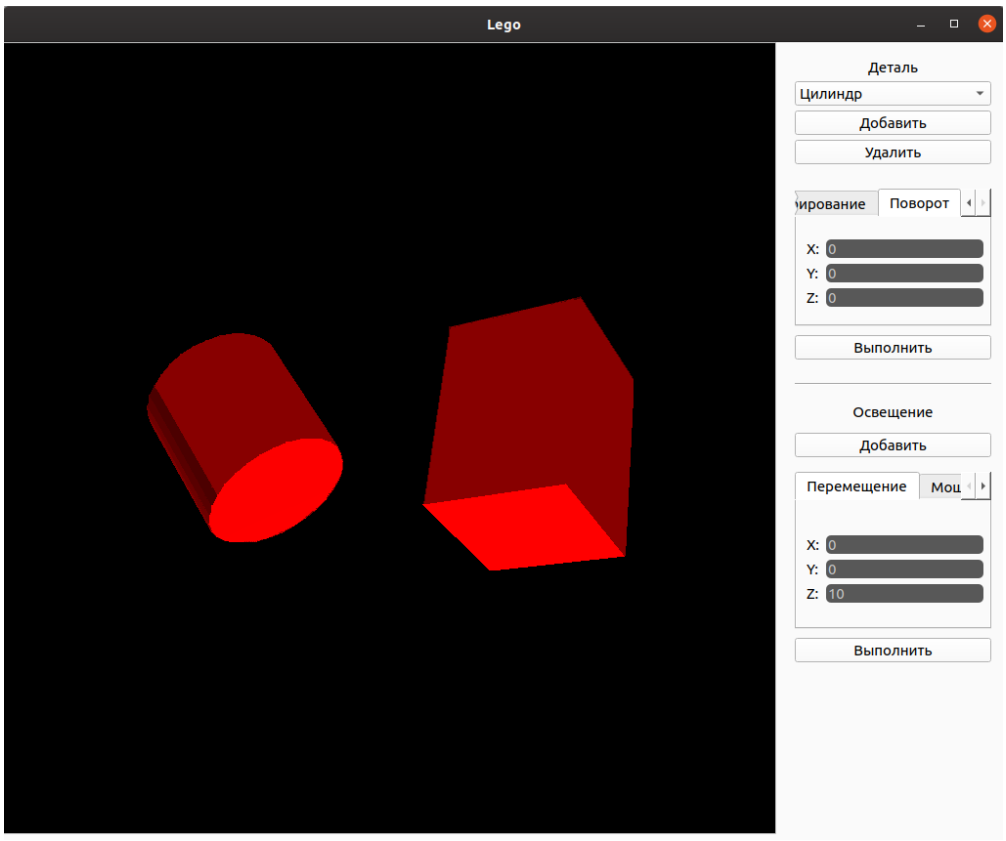


Рисунок 4.1.1. Пример работы приложения. Куб и цилиндр.

4.2. Постановка эксперимента.

Поскольку в проекте ведется работа с динамической сценой, требуется высокая скорость отрисовки сцены. Проведем исследование зависимости времени отрисовки от количества вершин объектов на сцене.

Замеры проводятся на объектах с количеством вершин 88, 264, 473, 1258, 2516, 3774. Во всех случаях на сцене расположен один источник освещения. Каждый замер производится по 10 раз для усреднения результата.

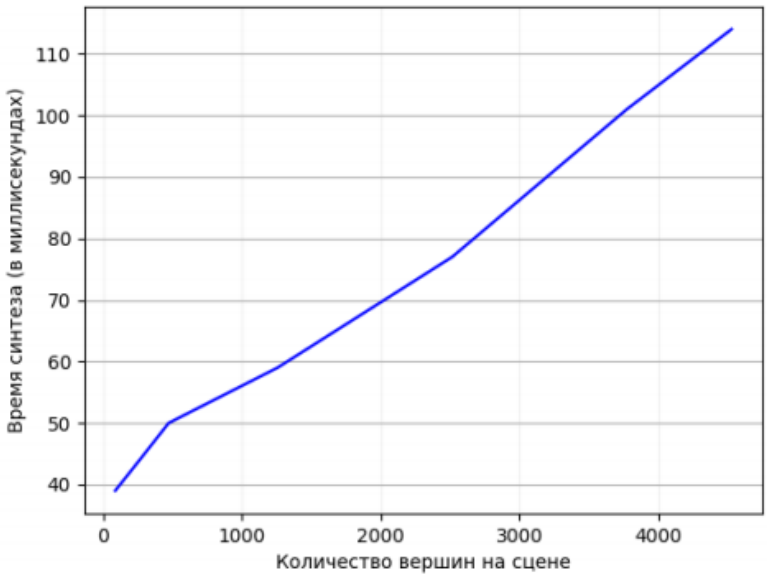
4.3. Результаты эксперимента.

Результаты эксперимента приведены в таблице 4.3.1 и на рисунке 4.3.2.

Таблица 4.3.1. Результаты замеров времени.

Количество вершин	88	264	473	1258	2516	3774	4528
Среднее время (мс)	39	44	50	59	77	101	114

Рисунок 4.3.2. График зависимости времени отрисовки сцены от количества вершин.



Вывод

В ходе эксперимента было установлено, что с увеличением количества вершин на сцене, время отрисовки значительно увеличивается (линейная зависимость). Также для фигур с 2516 вершинами и более снижается плавность при перемещении.

Заключение

Цель работы достигнута, все поставленные задачи выполнены: были разработаны и реализованы алгоритмы визуализации трёхмерных объектов, а также проведено тестирование и анализ их работы.

В ходе работы были проанализированы алгоритмы удаления невидимых линий и поверхностей, закраски, модели освещения, а также были указаны их достоинства, недостатки и применимость к поставленной задачи. Были разработаны собственные и модернизированы существующие структуры и классы, необходимые для решения поставленных задач.

В ходе выполнения проекта были усовершенствованы навыки работы с формами и были расширены знания в области компьютерной графики.

Литература

1. А. В. Куров. Лекции по курсу «Компьютерная графика».
2. Д. Роджерс. Алгоритмические основы машинной графики: Пер. с англ. – М.: Мир, 1989.-512с., ил.
3. А. Ю. Дёмин. Компьютерная графика / А. Ю. Дёмин, А. В. Кудинов – Режим доступа: http://www.compgraph.tpu.ru/index.html (дата обращения: 29.07.2020).
4. R. Szeliski. Computer Vision: Algorithms and Applications, 2nd ed., 2020
5. Е. А. Снижко. Компьютерная геометрия и графика [Текст], 2005. - 17 с.
6. Ю. Тихомиров. Программирование трехмерной графики – Спб: BHV – Санкт-Петербург, 1998 – 256с.: ил.