Оглавление

Введение

Компьютерную графику как науку можно назвать одной из самых распространенных и обширных сфер цифрового мира. Подавляющее число областей жизни человека, связанных с компьютером, так или иначе пересекаются с компьютерной графикой. Одна из самых очевидных направлений для применения компьютерной графики – фотография.

Получение изображений близких по своим характеристикам к «живым» картинам – одна из сложнейших, но, в то же время, одна из наиболее актуальных задач компьютерной графики. Одной из причин сложности генерации реалистичных изображений является необходимость учитывать сложные принципы распространения света.

Целью данной работы является выбрать алгоритмы для эффективной визуализации студийного освещения, а также обосновать этот выбор. Реализовать данные алгоритмы, а также, по возможности, адаптировать их к условиям задачи.

1 Аналитическая часть

В данной главе были рассмотрены объекты сцены, формы и способы задания трехмерных моделей, алгоритмы удаления невидимых ребер и поверхностей и модели освещения. На основании проведенного анализа были выбраны форма и способ задания трехмерных моделей, алгоритм удаления невидимых ребер и поверхностей и модель освещения, которые будут использоваться при реализации программы.

1.1 Описание объектов сцены

В данном разделе были описаны сущности, которые будут отображаться в процессе выполнения программы.

Сцена включает в себя:

- один или несколько источников света;
- одну или несколько фоновых плоскостей;
- тестируемую модель.

Источник света представляется в виде материальной точки, которая испускает лучи во все стороны. Если он устанавливается на «бесконечном» удалении от сцены, на сцену попадает множество параллельных лучей. В программе источники света будут играть роль студийных проекторов, однако визуализировать их не планируется.

Фоновые плоскости – условные границы сцены, которые также будут отображаться на итоговом изображении.

Тестируемая модель – некоторая объемная модель, которая будет использоваться для предпросмотра эффективности студийного освещения. В качестве тестируемой модели могут выступать как примитивы (кубы или пирамиды), так и более сложные модели (каркасы людей или животных).

1.2 Анализ и выбор формы задания трехмерных моделей

В данном разделе были рассмотрены формы задания трехмерных моделей. На основании проведенного анализа была выбрана одна из форм, которая будет использоваться при реализации программы.

В данной работе под моделью понимается отображение некоторых физических свойств объектов реального мира (размера и формы) в цифровом пространстве.

Выделяют три основных вида моделей:

- каркасная модель;
- поверхностная модель;
- объемная модель.

Рассмотрим основные характеристики этих моделей [1].

1.2.1 Каркасная модель

Каркасная модель — простейшая из представленных моделей, объекты задаются множеством точек в пространстве и связями между этими точками.

Основным преимуществом каркасной модели можно назвать простоту, к недостатком отнесем то, что такие модели порой бывает невозможно отобразить без искажения восприятия форм и размеров.

1.2.2 Поверхностная модель

Поверхностная модель [2] — как следует из названия, в этой модели объекты задаются в виде набора плоскостей.

Эта модель позволяет с достаточной точностью передать физические характеристики объекта, однако, если объект становится слишком сложным, могут возникнуть трудности с определением расположения материала внутри модели.

1.2.3 Объемная модель

Объемная модель — схожа с поверхностной моделью, однако добавляется список внутренних нормалей, благодаря которому мы можем однозначно понять где находится материал модели.

Такая модель хранит исчерпывающую информацию об объекте, что хорошо при работе со сложными сценами, однако зачастую излишне.

Вывод

Подводя итоги, можно сделать вывод, что оптимальным решением будет использование поверхностной модели, так как сложность предполагаемой сцены не требует хранения дополнительной информации об ориентированности плоскостей.

1.3 Анализ способа задания поверхностных моделей

В данном разделе были рассмотрены способы задания поверхностных моделей. На основании проведенного анализа была выбрана одна из них, который будет использоваться при реализации программы.

Существует два распространённых способа задания поверхностных моделей: аналитический и полигональный.

1.3.1 Аналитический способ

При аналитическом способе хранения поверхностных моделей возникают дополнительные ресурсные траты на получение фактических значений вершин полигонов модели;

1.3.2 Полигональный способ

При полигональном способе хранения не возникают трудности при получении фактических координат вершин полигонов модели, что может облегчить написание алгоритмов.

Возможные способы хранения информации о сетке:

1. Список граней.

   Объект – это множество граней и вершин. В каждую грань входят как минимум 3 вершины.

2. «Крылатое» представление.

   Каждая точка ребра указывает на две вершины, две грани и четыре ребра, которые её касаются.

3. Полурёберные сетки.

   То же «крылатое» представление, но информация обхода хранится для половины грани.

4. Таблица углов.

   Таблица, хранящая вершины. Обход заданной таблицы неявно задаёт 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.

5. Вершинное представление.

Хранятся лишь вершины, которые указывают на другие вершины. Простота представления даёт возможность проводить над сеткой множество операций.

Вывод

Было принято решение при реализации итогового продукта использовать полигональную сетку в виде списка полигонов из трех вершин. Такой подход позволит эффективно описывать сложные модели, при этом преобразования будут применяться достаточно просто и быстро. Так же, явное описание вершин модели упрощает написание алгоритма удаления невидимых граней.

1.4 Анализ и выбор алгоритма удаления невидимых ребер и поверхностей

В данном разделе были рассмотрены алгоритмы удаления невидимых ребер и поверхностей. На основании проведенного анализа был выбран один из них, который будет использоваться при реализации программы.

Выдвигаемые требования к свойствам алгоритма для обеспечения реалистичного изображения.

1. Алгоритм может работать как в объектном пространстве, так и в пространстве изображений.
2. Алгоритм должен быть достаточно быстрым.
3. Алгоритм должен иметь высокую реалистичность изображения.

1.4.1 Алгоритм, использующий Z-буфер

Суть данного алгоритма – это использование двух буферов: буфера кадра, в котором хранятся атрибуты каждого пикселя, и Z-буфера, в котором хранятся информация о координате Z для каждого пикселя [3].

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.

Преимущества алгоритма — простота реализации и оценка трудоемкости линейна. Недостатки алгоритма — сложная реализация прозрачности и большой объем требуемой памяти.

Данный алгоритм отвечает заявленным требованиям. Сложная реализация прозрачности, в данном случае, не существенна, так как изначально прозрачные объекты не планировались к рассмотрению.

1.4.2 Алгоритм обратной трассировки лучей

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

Преимущества алгоритма:

- высокая реалистичность синтезируемого изображения;
- работа с поверхностями в математической форме;
- вычислительная сложность слабо зависит от сложности сцены.

Недостатки алгоритма — алгоритм имеет низкую производительность.

1.4.3 Алгоритм Робертса

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

Алгоритм выполняется в 3 этапа.

1. Этап подготовки исходных данных.

   На данном этапе должна быть задана информация о телах. Для каждого тела сцены должна быть сформирована 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.

   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.

2. Этап удаления рёбер, экранируемых самим телом.

   На данном этапе рассматривается вектор взгляда $E = 0, 0, -1, 0$. Для определения невидимых граней достаточно умножить вектор E на матрицу тела V. Отрицательные компоненты полученного вектора будут соответствовать невидимым граням.

3. Этап удаления невидимых рёбер, экранируемых другими телами сцены.

   На данном этапе для определения невидимых точек ребра требуется построить луч, соединяющий точку наблюдения с точкой на ребре. Точка будет невидимой, если луч на своём пути встречает в качестве преграды рассматриваемое тело.

Преимущества алгоритма — работа в объектном пространстве и высокая точность вычисления.

Недостатки алгоритма:

- рост сложности алгоритма — квадрат числа объектов;
- тела сцены должны быть выпуклыми (усложнение алгоритма, так как нужна будет проверка на выпуклость);
- сложность реализации.

Алгоритм Робертса не подходит для решения поставленной задачи из-за высокой сложности реализации как самого алгоритма, так и его модификаций.

1.4.4 Алгоритм художника

Данный алгоритм работает аналогично тому, как художник рисует картину — то есть сначала рисуются дальние объекты, а затем более близкие. Наиболее распространенная реализация алгоритма — сортировка по глубине, которая заключается в том, что произвольное множество граней сортируется по ближнему расстоянию от наблюдателя, а затем отсортированные грани выводятся на экран в порядке от самой дальней до самой ближней. Данный метод работает лучше для построения сцен, в которых отсутствуют пересекающиеся грани.

Преимущество алгоритма — требуем меньше памяти, чем, например, алгоритм Z-буффера.

Недостатки алгоритма — недостаточно высока реалистичность изображения и сложность реализации при пересечения граней на сцене.

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

1.4.5 Алгоритм Варнока

Алгоритм Варнока является одним из примеров алгоритма, основанного на разбиении картинной плоскости на части, для каждой из которых исходная задача может быть решена достаточно просто.

Поскольку алгоритм Варнока нацелен на обработку картинки, он работает в пространстве изображения. В пространстве изображения рассматривается окно и решается вопрос о том, пусто ли оно, или его содержимое достаточно просто для визуализации. Если это не так, то окно разбивается на фрагменты до тех пор, пока содержимое фрагмента не станет достаточно простым для визуализации или его размер не достигнет требуемого предела разрешения.

Сравнивая область с проекциями всех граней, можно выделить случаи, когда изображение, получающееся в рассматриваемой области, определяется сразу:

- проекция ни одной грани не попадает в область;
- проекция только одной грани содержится в области или пересекает область. В этом случае проекции грани разбивают всю область на две части, одна из которых соответствует этой проекции;
- существует грань, проекция которой полностью накрывает данную область, и эта грань расположена к картинной плоскости ближе, чем все остальные грани, проекции которых пересекают данную область. В данном случае область соответствует этой грани.

Если ни один из рассмотренных трех случаев не имеет места, то снова разбиваем область на четыре равные части и проверяем выполнение этих условий для каждой из частей. Те части, для которых таким образом не удалось установить видимость, разбиваем снова и т.д.

Преимущество алгоритма — меньшие затраты по времени в случае области, содержащий мало информации.

Недостатки алгоритма — алгоритм работает только в пространстве изображений и большие затраты по времени в случае области с высоким информационным содержимым.

Данный алгоритм не отвечает требованию работы как в объектном пространстве, так и в пространстве изображений, а также возможны большие затраты по времени работы.

Вывод

Для решения поставленной задачи был выбран алгоритм использующий Z-буфер. Данный алгоритм позволит получить приемлемую реалистичность при достаточно простой реализации. В данном проекте, единственным недостатком этого алгоритма можно назвать большие затраты памяти, однако эта проблема не столь существенна на современных компьютерах.

1.5 Анализ и выбор модели освещения

В данном разделе были рассмотрены модели освещения. На основании проведенного анализа была выбрана одина из них, который будет использоваться при реализации программы.

Рассмотрим две модели освещения, а именно: модель Ламберта и модель 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) (Рисунок 1.1).

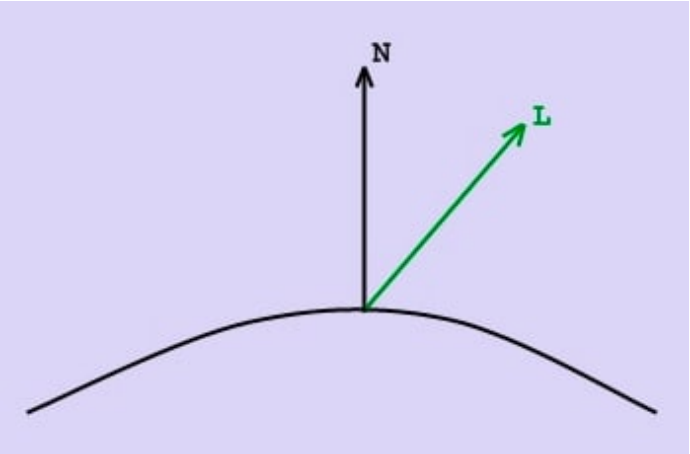

Рисунок 1.1 – Направленность источника света

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.

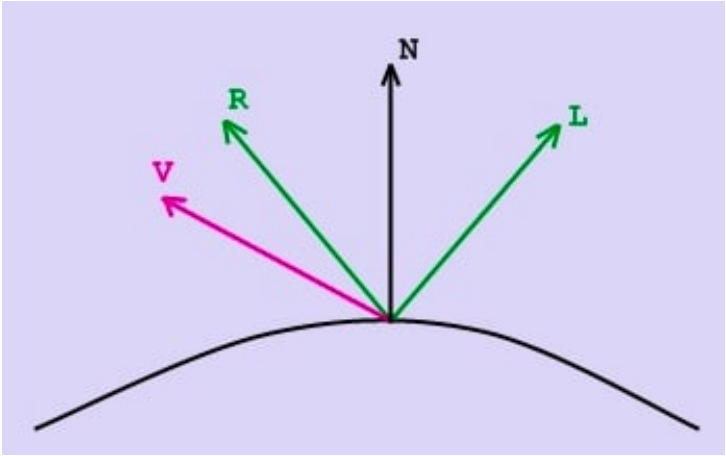

Рисунок 1.2 – Направление источника света, отраженного луча и наблюдателя

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.

Вывод

Модель Ламберта выглядит более эффективной с точки зрения производительности. В рамках поставленной задачи она является достаточной. Огромным плюсом также служит тот факт, что работа программы будет несколько быстрее, чем при использовании модели Фонга. Таким образом было принято решение использовать модель Ламберта.

Вывод

Для реализации итогового программного продукта было принято решение использовать поверхностную модель для описания объектов сцены, для хранения поверхностей использовать полигональную сетку (каждый полигон — три узла). В качестве алгоритма удаления невидимых граней и поверхностей был выбран алгоритм, использующий Z-буфер. В качестве модели освещения была выбрана модель Ламберта.

2 Конструкторская часть

В данной главе были детально рассмотрены все этапы работы программы, а также разработаны схемы некоторых алгоритмов, которые используются в программе.

2.1 Общий алгоритм решения поставленной задачи

Общий алгоритм решения поставленной задачи выглядит следующим образом:

- задать объекты сцены (фоновые плоскости, источники света, тестируемый объект);
- рассчитать освещенность поверхностей сцены;
- изобразить сцену.

2.2 Алгоритм использующий Z-буфер

Для визуализации сцены был выбран алгоритм построчного сканирования с Z-буфером.

Концепция Z-буфера является обобщением идеи буфера кадра, который используется для запоминания атрибутов пикселей в пространстве кадра, в то время как Z-буфер хранит единственный атрибут – глубину пиксела относительно картинной плоскости. (Картинная плоскость, исторически, помещается в плоскость XOY, поэтому перпендикулярное этой плоскости направление, как глубина, легло в название буфера.)

Изначально Z-буфер инициализируется максимальным значение глубины для данной системы объектов или «бесконечностью». Задача алгоритма, далее, сводится к банальному поиску минимального значения глубины для каждого пиксела и отображению его на экран с учетом характеристик объекта, к которому он принадлежит.

На рисунке 2.1 представлена схема алгоритма использующего Z-буфер.

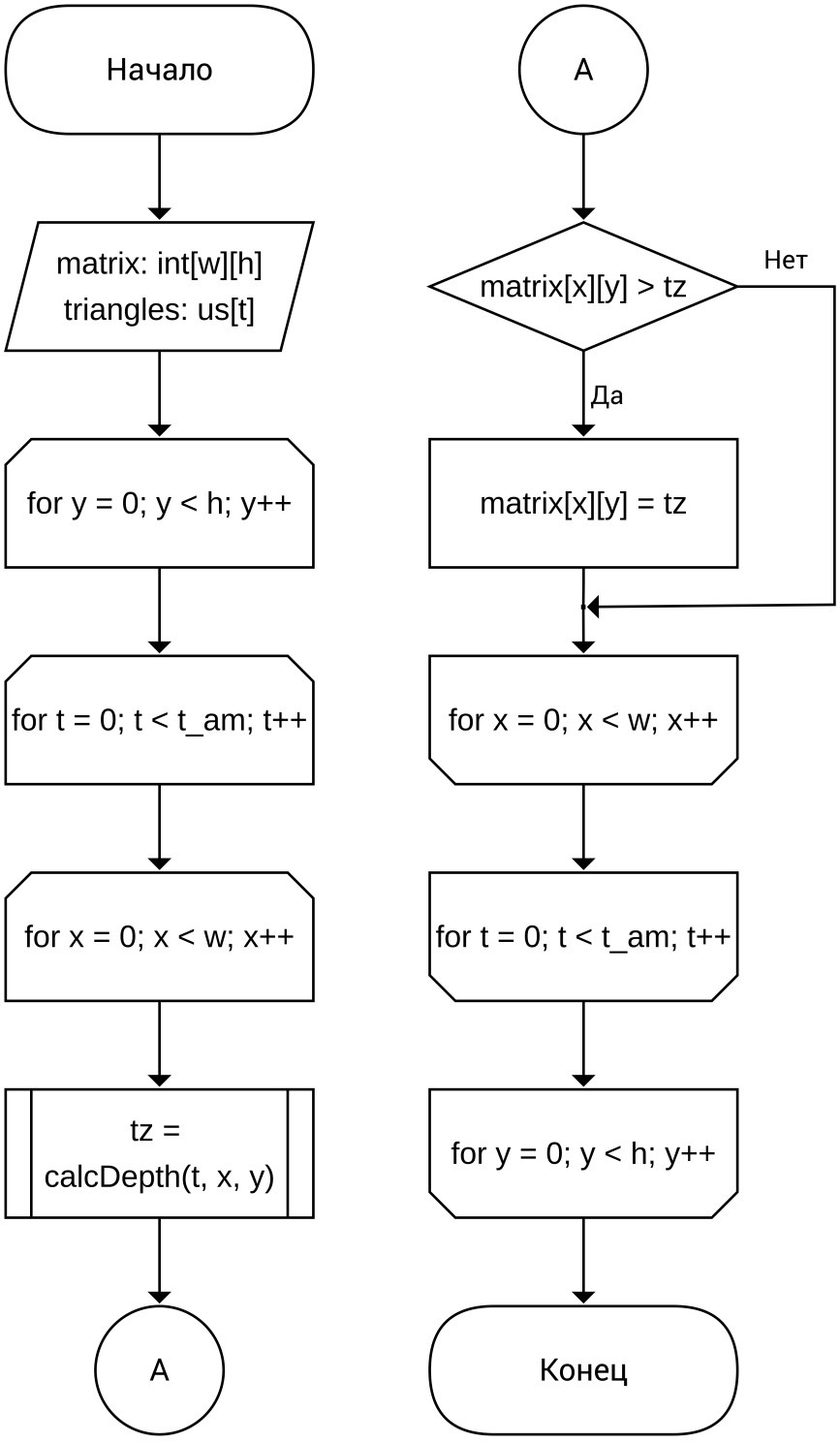

Рисунок 2.1 – Схема алгоритма удаления невидимых граней и поверхностей, использующего Z-буфер (последовательная версия)

В данном случае, количество граней полигона строго фиксируется значением три. «Преимущества данного представления: простое разбиение массива точек на треугольники, обеспечение хорошей реалистичности отоб-

ражения поверхности воды вследствие возможности детального моделирования рельефной поверхности, удобство в использовании треугольников в алгоритме построчного сканирования, использующего Z-буфер, из-за сведения оперирования многоугольниками к оперированию треугольниками.» — К.А. Якиль.

Недостатки алгоритма — большой объем требуемой для работы памяти.

При работе с данным алгоритмом приходится столкнуться с трудоемкостью реализации эффектов, связанных с полупрозрачностью, и ряда других специальных задач, повышающих реалистичность изображения. Поскольку алгоритм заносит пиксели в буфер кадра в произвольном порядке, то довольно сложно получить информацию, которая необходима для методов, основывающихся на предварительном анализе сцены.

Преимущества алгоритма — алгоритм достаточно «простой» для понимания и реализации и алгоритм решает задачу не только удаления невидимых поверхностей, но и визуализации пересечения поверхностей.

Большой объем требуемой памяти этого алгоритма частично исправляется тем фактом, что он будет использоваться в построчном сканировании, а значит объем выделяемой памяти будет соизмерим с шириной экрана, а не с его площадью, как это описано в классической реализации.

Реализация эффектов полупрозрачности не ставится основным требованием для данной работы, а лишь служит возможным ее улучшением, что дает нам возможность не рассматривать эту проблему.

2.3 Основной цикл программы

После выполнения просчета освещенности поверхностей наступает этап последовательной отрисовки сцены под разными углами, которые задаются пользователем.

Общий алгоритм работы программы:

- рассчитать необходимые углы поворота сцены;
- для каждого угла, отрисовать сцену, повернутую на значение этого угла;

- для каждого угла, сохранить получившееся изображение.

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.

2.5 Генерация тестируемой модели

Тестируемую модель задает пользователь. Тестируемая модель состоит из источников освещения и графических примитивов (кубов и шаров).

Вывод

В данной главе были детально рассмотрены все этапы работы программы, а также разработаны схемы некоторых алгоритмов, которые используются в программе.

3 Технологическая часть

В данной главе был выбран язык программирования, описана структура классов программы и разработана ее диаграмма, был описан интерфейс готовой программы и приведены примеры работы программы.

3.1 Выбор языка программирования

Для реализации данной программы был выбран язык программирования C/C++ [4].

Следующие факторы повлияли на этот выбор.

1. Выбранный язык программирования предоставляет широкие возможности для управления ресурсами (в первую очередь выделяемой памятью).

2. Данный язык является строго типизированным. Этот факт облегчает тестирование.

3. Программа на языке программирования C/C++ не обязана иметь фиксированную структуру (весь код, теоретически, может быть помещен в один файл).

3.2 Структура классов программы

При реализации данного программного обеспечения было выделено несколько математических абстракций, которые были описаны пользовательскими классами:

- *Object* – абстракция базового объекта;
- $Vertex$ – абстракция вершины;
- $Vertexes$ – абстракция множества связанных вершин;
- $Triangle$ – формализация связи трех вершин;

- *Triangles* – множество связанных формализаций;
- *Figure* – множество связанных вершин и формализаций, образующее обособленную «фигуру»;
- *LightSource* – абстракция источника света;
- *Camera* – абстракция камеры;
- *Scene* – множество обособленных «фигур».

3.3 Диаграмма классов

Диаграмма классов данного проекта представлена на рисунке 3.1.

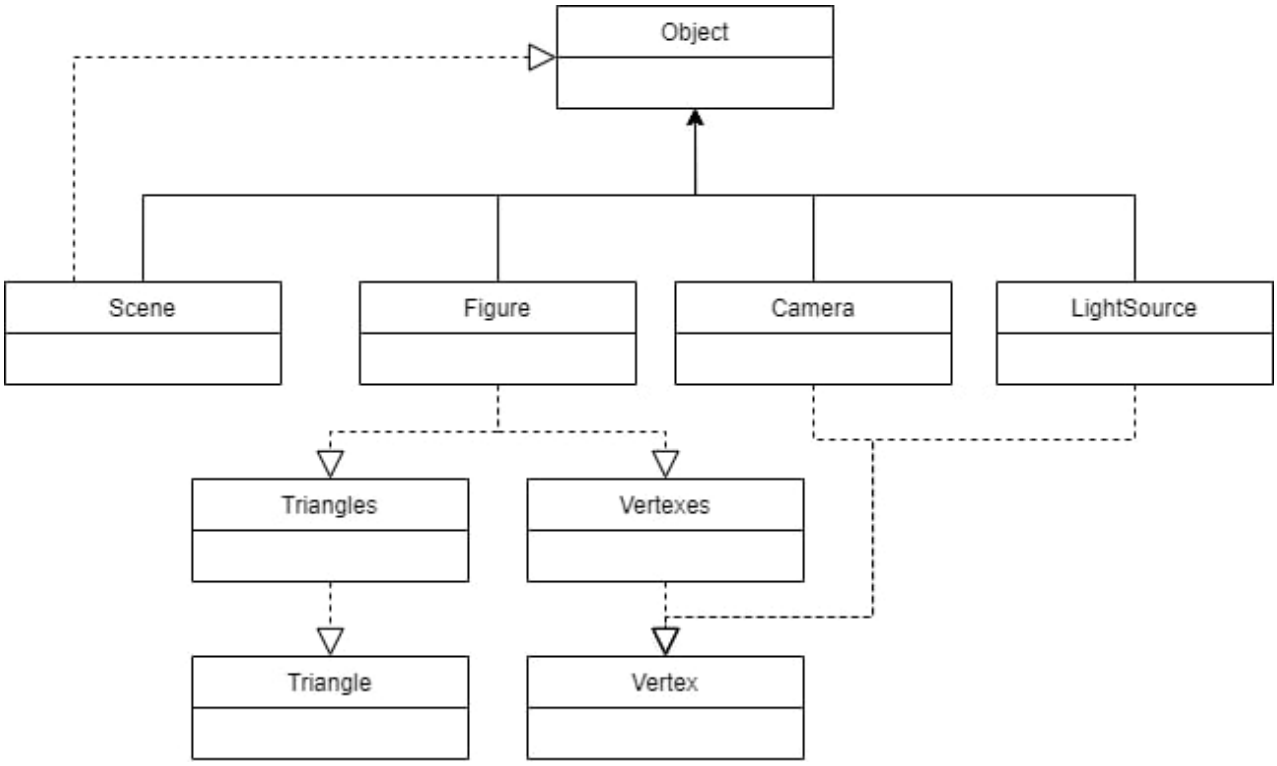

Рисунок 3.1 – Диаграмма классов

3.4 Интерфейс программного обеспечения

Основные возможности предоставляемые пользователю со стороны интерфейса:

- добавление графического примитива (куб, шар);
- добавление источника света;
- поворот сцены;
- смещение сцены;
- масштабирование сцены.

Изображение интерфейса программы представлено на рисунке 3.2.

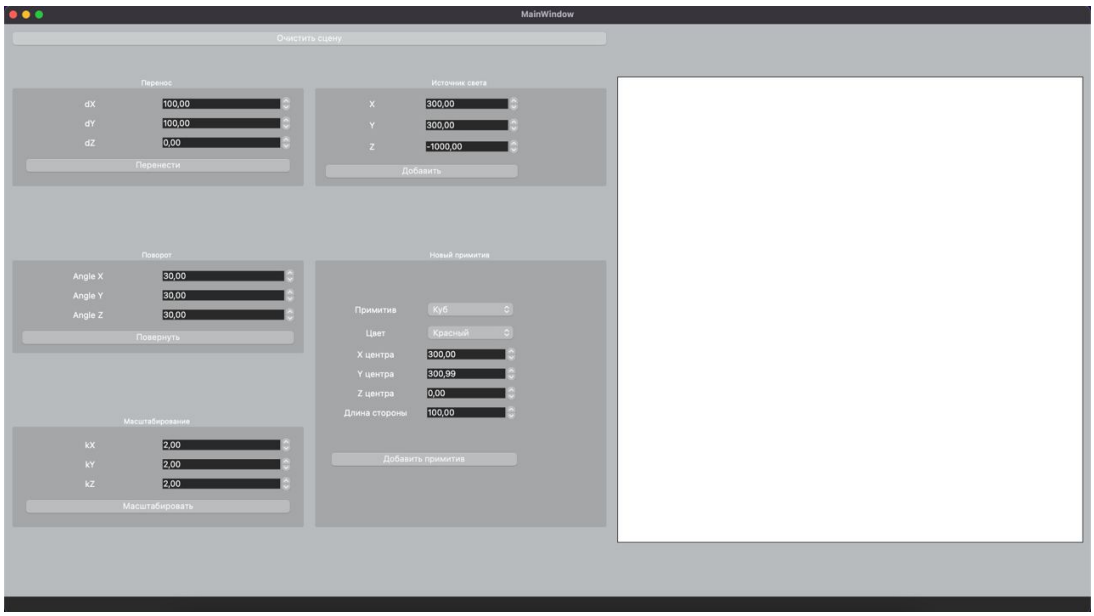

Рисунок 3.2 – Интерфейс программы

3.5 Результаты работы программного обеспечения

В данном разделе представлены изображения результатов работы программы (отдельных ее частей и всего интерфейса целиком).

На рисунке 3.3 представлено изображение шара, сгенерированного с малой степенью детализации.

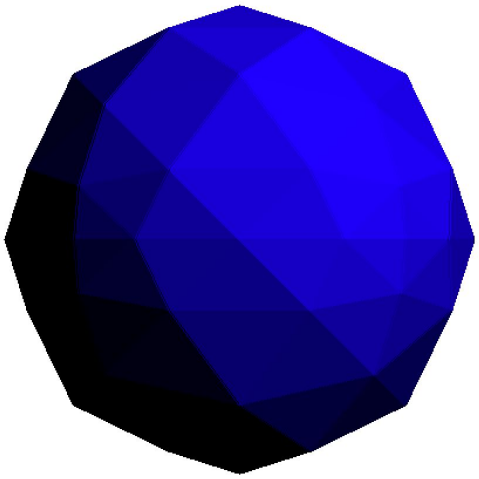

Рисунок 3.3 – Малая степень детализации шара

На рисунке 3.4 представлено изображение шара, сгенерированного с средней степенью детализации.

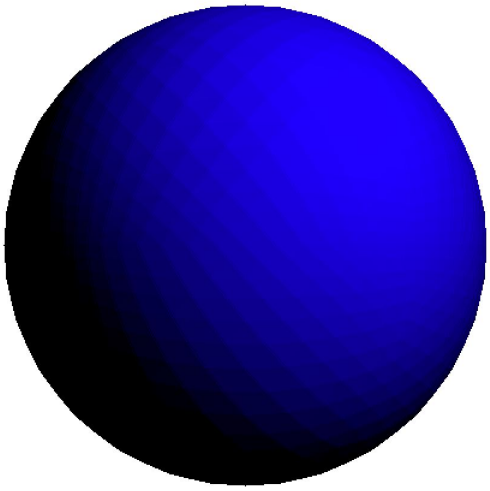

Рисунок 3.4 – Средняя детализация шара

На рисунке 3.5 представлено изображение шара, сгенерированного с высокой степенью детализации.

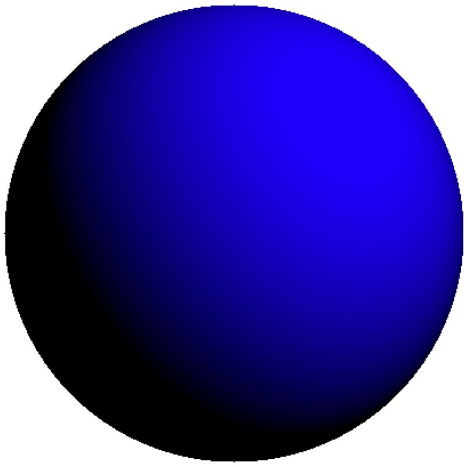

Рисунок 3.5 – Высока степень детализации шара

Из представленных изображений видно, что с повышением детализации модели, недостатки модели освещения Ламберта становятся менее заметными.

На рисунке 3.6 представлен интерфейс программы во время выполнения. В данном примере был создан один источник света, четыре шара разных цветов.

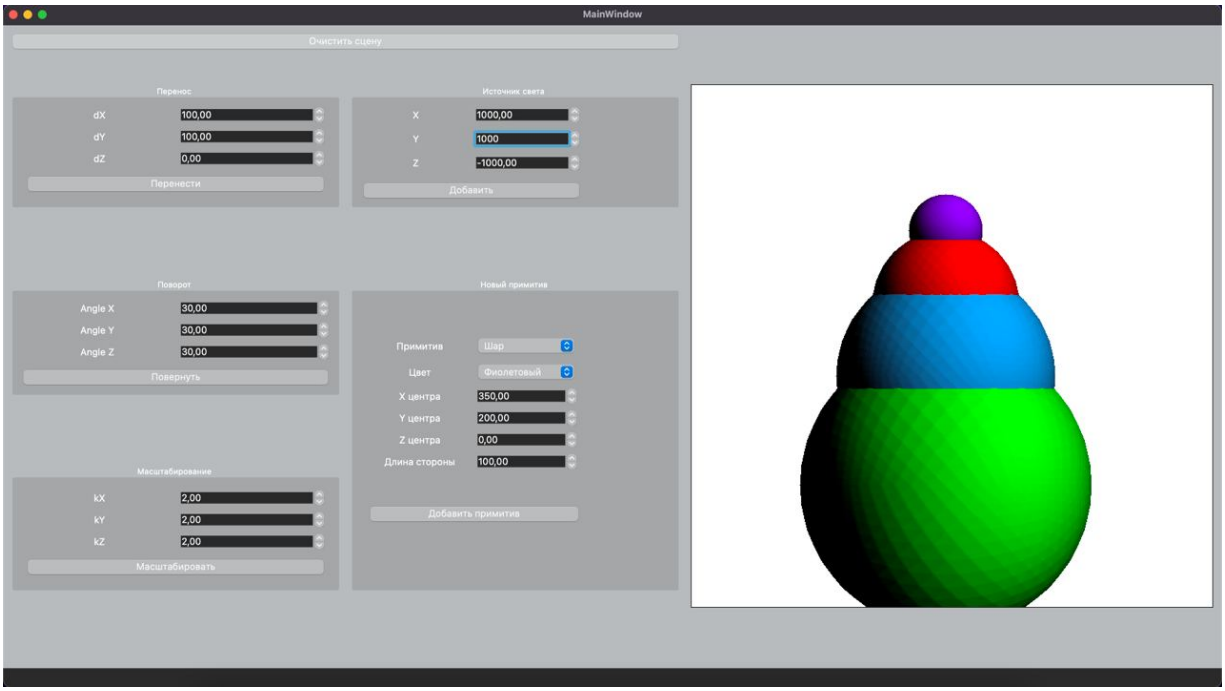

Рисунок 3.6 – Интерфейс программы во время выполнения

Вывод

В этом разделе был выбран язык программирования, рассмотрена uml-диаграмма основных классов, подробно разобран интерфейс приложения и приведены результаты работы програмы.

4 Исследовательская часть

В данном разделе были приведены примеры работы программ, постановка эксперимента и сравнительный анализ алгоритмов на основе полученных данных.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование, следующие:

- Операционная система Ubuntu 22.04.1 [5] Linux x86_64.
- Память 8 ГБ.
- Процессор AMD® Ryzen 5 3500u.

Тестирование проводилось на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, а также непосредственно системой тестирования.

4.2 Время выполнения алгоритмов

Для замера процессорного времени использовалась функция $std :: chrono :: system\_clock :: now(...)$ из библиотеки *chrono* языка программирования $C++$. Функция возвращает процессорное время типа $float$ в секун- дах.

Замеры времени для каждой длины входного массива треугольников проводились 1000 раз. В качестве результата взято среднее время работы алгоритма на данной длине слова. При каждом запуске алгоритма, на вход подавались случайно сгенерированные массивы треугольников. Тестовые пакеты создавались до начала замера времени.

4.2.1 Время выполнения однопоточной реализации

Результаты замеров приведены в таблице 4.1.

Обозначения: ОВ — однопоточное выполнение (C/C++), ПВ - последовательное выполнение (C/C++), МВ - многопроцессное выполнение (Python), КТ - количество треугольников.

Таблица 4.1 – Результаты замеров времени ОВ, ПВ, МВ для разных КТ (в микросекундах).

КТ	ОВ	ПВ	МВ
10	30	28	1832
20	35	31	2798
30	40	36	5125
50	43	39	9782
100	57	49	24320
200	75	69	56680

4.2.2 Время выполнения многопоточной реализации

Данное исследование проводилось вокруг реализации на ЯП C/C++, так как ЯП Python3 не поддерживает нативные потоки.

Результаты замеров приведены в таблице 4.2.

Таблица 4.2 – Время выполнения программы, реализующей многопоточный алгоритм удаления невидимых граней, использующий Z-буфер, для разного количества потоков (в микросекундах).

Количество треугольников	Количество потоков					
	1	4	6	12	24	48
10	30	27	33	34	41	47
100	57	75	76	76	81	88
1000	245	143	118	121	133	178
10000	2098	845	688	788	822	999
100000	19935	5606	5231	6300	7653	9923

При 6 потоках достигается пик, при котором все логические ядра процессора одновременно выполняют параллельные ветки алгоритма. Далее при увеличении числа потоков производительность падает. Это объясняется тем, что создается очередь потоков, которая замедляет работу программы.

Вывод

В данном разделе было произведено сравнение времени выполнения реализации алгоритма удаления невидимых ребер, при последовательной реализации и многопоточной. Результат показал, что выгоднее всего по времени использовать столько потоков, сколько у процессора логических ядер.

Заключение

В рамках данного курсового проекта были:

- описаны структуры трехмерной сцены, включая объекты, из которых состоит сцена;
- проанализированы и выбраны необходимые существующие алгоритмы для построения сцены;
- проанализированы и выбраны варианты оптимизации ранее выбранного алгоритма удаления невидимых линий;
- реализованы выбранные алгоритмы;
- разработано программное обеспечение, которое позволит отобразить трехмерную сцену;
- проведены сравнение стандартного и реализованного оптимизированного алгоритма удаления невидимых линий.

В ходы выполнения эксперимента было показано, что многопоточная реализация алгоритма удаления невидимых граней, использующего Z-буфер, может показывать меньшее время выполнения, чем однопоточная или линейная реализация. Также было проведено сравнение времени выполнения программ реализующих данный алгоритм на языках программирования C/C++ и Python.

Список использованных источников

[1] Н.Н. Голованов. Геометрическое моделирование. Физико-математической литературы, 2002. с. 472.

[2] Ю.Н. Косников. Поверхностные модели в системах трехмерной компьютерной графики. Учебное пособие. Пенза: Пензенский государственный университет, 2007. с. 60.

[3] П.В.Вельтмандер. Учебное пособие 'Основные алгоритмы компьютерной графики'. 1997.

[4] Рэнди Дэвис Стефан. C++ для чайников. Для чайников. Вильямс, 2018. с. 400.

[5] Ubuntu 20.04.3 LTS (Focal Fossa) [Электронный ресурс]. Режим доступа: `https://releases.ubuntu.com/20.04/`.