Оглавление

Введение

Компьютерная графика – это раздел программирования, предметом изучения которого являются методы создания реалистичных изображений. Алгоритмы, лежащие в основе этих методов, являются крайне затратными по времени, поскольку требуется обрабатывать модели, состоящие из огромного количества геометрических примитивов. Благодаря росту вычислительной производительности современных компьютеров синтез изображений всё чаще находит применение в системах автоматизированного проектирования, дизайне, системах виртуальной реальности, научной визуализации, медицине, индустрии электронных развлечений, кинематографе и мультипликации.

Одной из фундаментальных задач компьютерной графики является визуализация поверхностей. Решения этой задачи находят широкое применение при моделировании различных деталей и конструкций, для пакетов программ математической обработки.

Целью данного курсового проекта является разработка программы визуализации поверхностей, заданных функциями от двух переменных.

Задачи работы

В рамках выполнения работы необходимо решить следующие задачи:

- изучение и анализ существующих алгоритмов компьютерной графики, которые используются для создания реалистичной модели взаимно перекрывающихся объектов, и выбор наиболее подходящих для решения поставленной задачи;
- проектирования архитектуры программы и её интерфейса;
- реализация выбранных алгоритмов и структур данных;
- проведение исследования на основе разработанной программы.

1 Аналитический раздел

1.1 Анализ алгоритмов удаления невидимых линий и поверхностей

Одной из основных задач при построении реалистичного изображения является задача удаления объектов или их частей, которые перекрываются другими объектами, то есть являются невидимыми с точки зрения наблюдателя.

Для решения этой задачи выделяют две группы алгоритмов [1].

- Алгоритмы, работающие в объектном пространстве. Данные алгоритмы имеют привязку к мировой или физической системе координат. Получаемые результаты ограничиваются только точностью вычислений, однако требуют большого объема вычислений, зависящего от требуемой точности и сложности поступающей на вход сцены. В эту группу входят алгоритм Робертса, алгоритм со списком приоритетов и т. д.

- Алгоритмы, работающие в пространстве изображения. Данные алгоритмы предполагают привязку к системе координат экрана или картинной плоскости, на которую производится проецирование изображаемых объектов. Объем требуемых вычислений значительно меньше, чем у алгоритмов первой группы, и зависит от разрешающей способности экрана и количества объектов на сцене. Основными представителями данной группы являются алгоритм Варнока, алгоритм, использующий Z-буфер, и алгоритм трассировки лучей.

Для выбора наиболее подходящего для достижения поставленных задач алгоритма из перечисленных, необходимо осуществить их краткий обзор, отобрать критерии для сравнения и выявить алгоритм, который удовлетворяет этим критериям.

1.1.1 Алгоритм Робертса

Алгоритм Робертса представляет собой первое известное решение задачи об удалении невидимых линий. Данный алгоритм работает в объектном пространстве. В соответствии с алгоритмом, предварительно из каждого тела удаляются грани, экранируемые самим телом. Алгоритм Робертса сравнивает каждое из видимых ребер с каждым оставшимся объектом сцены для определения, какая часть или части, если таковые имеются, экранируются этими объектами.

К преимуществам данного алгоритма можно отнести тот факт, что математические методы, используемые в нем, просты, мощны и точны. Более поздние реализации алгоритма, например использующие предварительную сортировку вдоль оси z, демонстрируют почти линейную зависимость от числа объектов.

Недостатком алгоритма Робертса является то, что его вычислительная трудоемкость растет как квадрат числа объектов на сцене. Реализация оптимизированных алгоритмов весьма сложна.

1.1.2 Алгоритм Варнока

Алгоритм Варнока работает в пространстве изображения. В основе этого алгоритма лежит использование свойства когерентности изображения, то есть однородности его смежных областей. [1][2]

Простой для визуализации будем считать некоторую область изображения, полигон в которой отсутствует или является единственным и не пересекает границы данной области (см. рис. 1.1). В пространстве изображения рассматривается область и проверяется, является ли оно достаточно простым для визуализации. Если это не так, то область изображения разделяется на подобласти до тех пор, пока каждая подобласть не станет достаточно простой для визуализации или ее размер не достигнет предела разрешения дисплея.

В противном же случае разбиение повторяется, и для каждой из вновь полученных подобластей рекурсивно применяется процедура принятия ре-

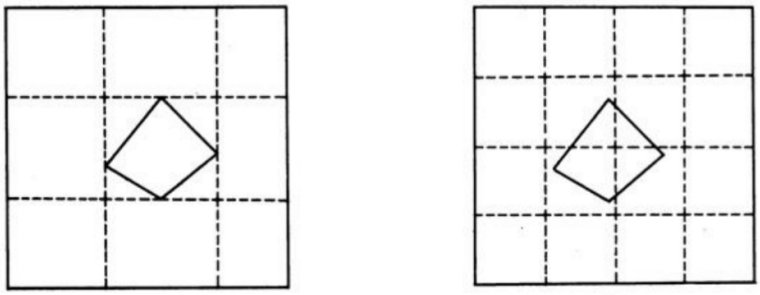

Рис. 1.1: Каждая подобласть является простой для визуализации (слева), некоторые подобласти не являются простыми для визуализации (справа)

шения. Предполагается, что с уменьшением размеров области ее перекрывает все меньшее и меньшее количество многоугольников. Считается, что в пределе будут получены области, содержащие не более одного многоугольника, и решение будет принято достаточно просто. Если же в процессе разбиения будут оставаться области, содержащие не один многоугольник, то следует продолжать процесс разбиения до тех пор, пока размер области не станет совпадать с одним пикселом. В этом случае для полученного пикселя необходимо вычислить глубину (значение координаты Z) каждого многоугольника и визуализировать тот из них, у которого максимальное значение этой координаты.

Достоинством данного алгоритма является простота реализации и высокая эффективность в случае, если размеры перекрываемых областей невелики.

Недостатком алгоритма Варнока является использование рекурсивных вызовов, что значительно снижает скорость выполнения в случае больших размеров перекрываемых областей.

1.1.3 Алгоритм, использующий Z-буфер

Данный алгоритм удаления невидимых поверхностей является одним из самых простых и широко используемых. Его идея заключается в использовании двух буферов: буфера кадра и буфера глубины, также называемого Z-буфером. Буфер кадра используется для хранения интенсивности каждого пикселя в пространстве изображения. В буфере глубины запоминается значение координаты Z (глубины) каждого видимого пикселя в простран-

стве изображения. В ходе работы алгоритма значение глубины каждого нового пикселя, заносимого в буфер кадра, сравнивается с глубиной того пикселя, который уже занесен в Z-буфер. Если это сравнение показывает, что новый пиксель расположен ближе к наблюдателю, чем пиксель, уже находящийся в буфере кадра, то новый пиксель заносится в буфер кадра и производится корректировка Z-буфера: в него заносится глубина нового пикселя. Если же значение глубины нового пикселя меньше, чем хранящееся в буфере, то осуществляется переход к следующей точке. [1][2]

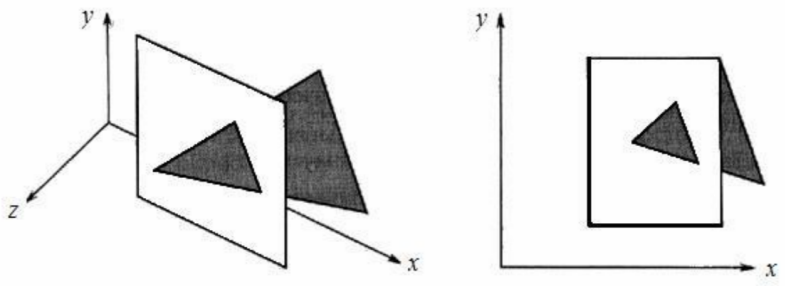

Рис. 1.2: Пример работы алгоритма z-буфера

Основными достоинствами данного алгоритма являются простота его реализации, корректная обработка случаев взаимных пересечений объектов, линейная зависимость трудоемкости от числа объектов на сцене, а также отсутствие необходимости предварительной сортировки объектов по глубине, то есть они могут обрабатываться в произвольном порядке.

К недостаткам данного алгоритма относят необходимость выделения памяти под два буфера, каждый из которых имеет размер равный количеству пикселей на экране.

1.1.4 Алгоритм трассировки лучей

В данном алгоритме из каждого пикселя картинной плоскости выходит луч, который пересекает ближайшую к наблюдателю грань, тем самым определяя ближайший объект, препятствующий его дальнейшему распространению (см рис. 1.3). Трассировка может прекращаться при пересечении лучом поверхности видимого непрозрачного объекта или применять более полные модели освещения с учетом эффектов отражения одного объекта от поверхности другого, преломления, прозрачности и затенения.

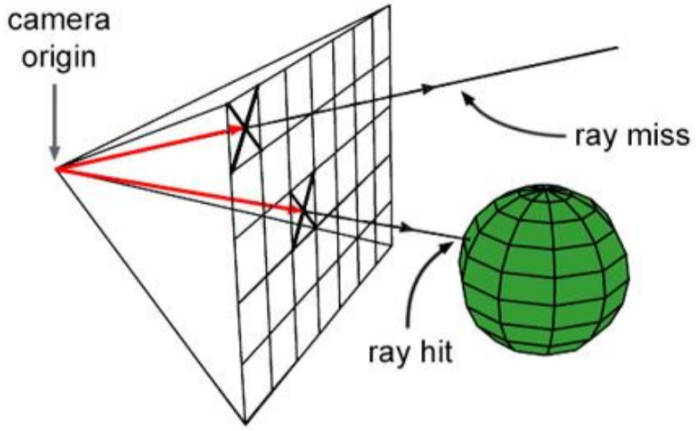

Рис. 1.3: Пример работы алгоритма трассировки лучей

К достоинствам данного алгоритма можно отнести возможность получения изображения гладких объектов без аппроксимации их примитивами (например, треугольниками). Вычислительная сложность метода линейно зависит от сложности сцены. Нетрудно реализовать наложение света и тени на объекты. Качество полученного изображения получается очень реалистичным, этот метод отлично подходит для создания фотореалистичных картин.

Серьёзным недостатком алгоритма трассировки лучей является производительность. Для получения изображения необходимо создавать огромное число лучей, проходящих через сцену и отражаемых от объекта. Это приводит к существенному снижению скорости работы программы.

1.1.5 Критерии выбора удаления невидимых линий и поверхностей

При реализации программы визуализации поверхностей необходим алгоритм удаления невидимых линий и поверхностей со следующими критериями:

1. высокая скорость работы (т. к. сцена должна меняться в реальном времени при повороте камеры):
   (a) линейная трудоёмкость от числа объектов сцены N;
   (b) без использования рекурсивных вызовов;

(c) возможность распараллеливания;

2. широкая распространённость в современном ПО.

1.1.6 Выбор алгоритма удаления невидимых линий и поверхностей

Факт задействования дополнительной памяти под буферы не является весомым критерием и может быть опущен, так как среднестатистический компьютер имеет 8 Гб оперативной памяти [3].

Наибольшее распространение в современном программном обеспечении получили алгоритмы трассировки лучей и Z-буфера: алгоритм Z-буфера используется в большинстве графических движков [4][5], а алгоритм трассировки лучей используется в ПО для создания фотореалистичных сцен [6].

Алгоритм, использующий Z-буфер, соответствует всем отобранным критериям.

1.2 Анализ алгоритмов закраски

Существуют три основных алгоритма, позволяющих закрасить полигональную модель: простая закраска, закраска по Гуро и закраска по Фонгу.

1.2.1 Простая модель освещения

Суть данного алгоритма заключается в том, что для каждой грани объекта находится вектор нормали, и с его помощью в соответствии с выбранной моделью освещения вычисляется значение интенсивности, с которой закрашивается вся грань. [7]

К достоинствам метода можно отнести производительность. Заметим, что все точки грани будут иметь одинаковую интенсивность, что является недостатком алгоритма.

1.2.2 Закраска по Гуро

Данный метод отличается от простой закраски тем, что разные точки грани закрашиваются с разными значениями интенсивности. Для это в каждой вершине грани находится вектор нормали и вычисляется значение интенсивности. Затем найденные значения интенсивности интерполируются по всем точкам грани. [7]

Закраска по Гуро позволяет получить реалистичное диффузное отражение. Если каждая плоская грань имеет один постоянный цвет, определенный с учетом отражения, то различные цвета соседних граней очень заметны, и поверхность выглядит именно как многогранник. Казалось бы, этот дефект можно замаскировать с помощью увеличения количества граней при аппроксимации поверхности, но человеческое зрение имеет способность подчеркивать перепады яркости на границах смежных граней – такой эффект называется эффектом полос Маха.

1.2.3 Закраска по Фонгу

Закраска Фонга по своей идее похожа на закраску Гуро, но ее отличие состоит в том, что в методе Гуро по всем точкам полигона интерполируются значения интенсивностей, а в методе Фонга – вектора нормалей, и с их помощью для каждой точки находится значения интенсивностей, а в методе Фонга – вектора нормалей, и с их помощью для каждой точки находится значение интенсивности. [7]

Данный алгоритм позволяет получить реалистичное зеркальное отражение. Хотя метод Фонга устраняет большинство недостатков предыдущего метода, он также основан на линейной интерполяции, поэтому в местах разрыва первой производной интенсивности заметен эффект полос Маха, однако менее выраженный, чем при закраске Гуро.

1.2.4 Выбор алгоритма закраски

На рисунке 1.4 наглядно показаны различия рассмотренных методов закраски.

Рис. 1.4: Методы закраски (слева направо: плоская, Гуро, Фонга)

Так как для удаления невидимых поверхностей мы выбрали Z-буфер, в котором используется интерполяция для вычисления глубины каждый точки, нам хорошо подойдёт закраска по Гуро, так же требующая интерполяцию. Совмещение алгоритма Z-буфера и алгоритма Гуро позволит сократить объем вычислений за счёт исключения этапа формирования промежуточного массива точек изображения, который является исходным для определения цвета каждой точки изображения.

Вывод

В результате проведенного анализа в качестве алгоритма удаления невидимых линий был выбран алгоритм, использующий Z-буфер, в качестве алгоритма закраски — алгоритм закраски по Гуро.

Предложен метод сокращения объемов вычислений за счёт исключения этапа формирования промежуточного массива точек изображения.

2 Конструкторский раздел

2.1 Описание структур данных

Для формализации общего алгоритма синтеза изображения в данной программе, необходимо ввести определения использующихся в ней структур данных:

- Сцена представляет собой список с произвольным числом моделей и объект камеры.
- Модель включает в себя следующие данные:
  - массив вершин фигуры;
  - массив рёбер фигуры;
  - массив полигонов фигуры;
  - цвет поверхности.
- Камера содержит:
  - положение в пространстве;
  - систему координат камеры, задаваемую тремя ортогональными векторами;

Вершины задаются координатами (x, y, z), ребро — двумя индексами вершин в массиве вершин, полигон — тремя индексами вершин в массиве вершин. Для последующей растеризации более удобный формат хранения полигона — треугольник, заданный сразу тремя вершинами.

2.2 Моделирование поверхности функции от двух переменных

Чтобы перейти от рассмотрения функции к рассмотрению поверхности, необходимо построить модель, то есть представить объект в виде полигонов, аппроксимирующих поверхность, заданную функцией. В данной

программе поверхности ограничены по двум координатам, соответственно, можно составить ортогональную сетку, в узлах которой будут значения функции. Так как ортогональная сетка представляется прямоугольниками, необходимо каждую ячейку сетки разделить на треугольники. Триангуляцию можно осуществить двумя путями: соединить вершины диагонали прямоугольника или выделить новую вершину в центре прямоугольника, соединив её со всеми вершинами этого прямоугольника (см. рис. 2.1).

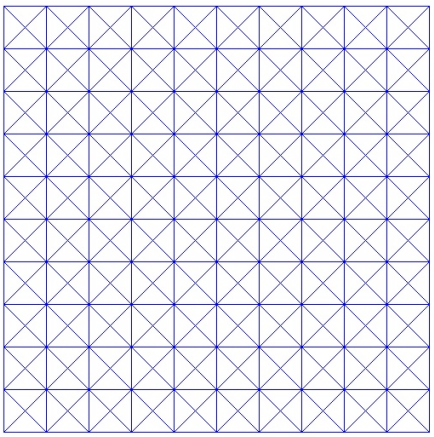

Рис. 2.1: Сетка поверхности

2.3 Аффинные преобразования

Для реализации возможности поворота и масштабирования относительно наблюдаемых поверхностей, необходимо воспользоваться матрицами аффинных преобразований [2].

Матрица масштабирования относительно начала координат с коэффициентами k_x, k_y, k_z:

$$\begin{pmatrix} \frac{1}{k_x} & 0 & 0 & 0 \\ 0 & \frac{1}{k_y} & 0 & 0 \\ 0 & 0 & \frac{1}{k_z} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.1}$$

Матрицы поворота на угол φ

- относительно оси oX:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\varphi & \sin\varphi & 0 \\ 0 & -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \tag{2.2}$$

- относительно оси oY:

$$\begin{pmatrix} \cos\varphi & 0 & \sin\varphi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\varphi & 0 & \cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \tag{2.3}$$

- относительно оси oZ:

$$\begin{pmatrix} \cos\varphi & \sin\varphi & 0 & 0 \\ -\sin\varphi & \cos\varphi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{2.4}$$

Матрица перенос вдоль координатных осей на dx, dy, dz:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ dx & dy & dz & 1 \end{pmatrix}. \tag{2.5}$$

2.4 Разработка алгоритма, объединяющего Z-буфер и закраску по Гуро

В аналитическом разделе был предложен метод сокращения объемов вычисления за счет исключения этапа формирования промежуточного массива точек. Схема предложенного алгоритма показана на рисунке 2.2.

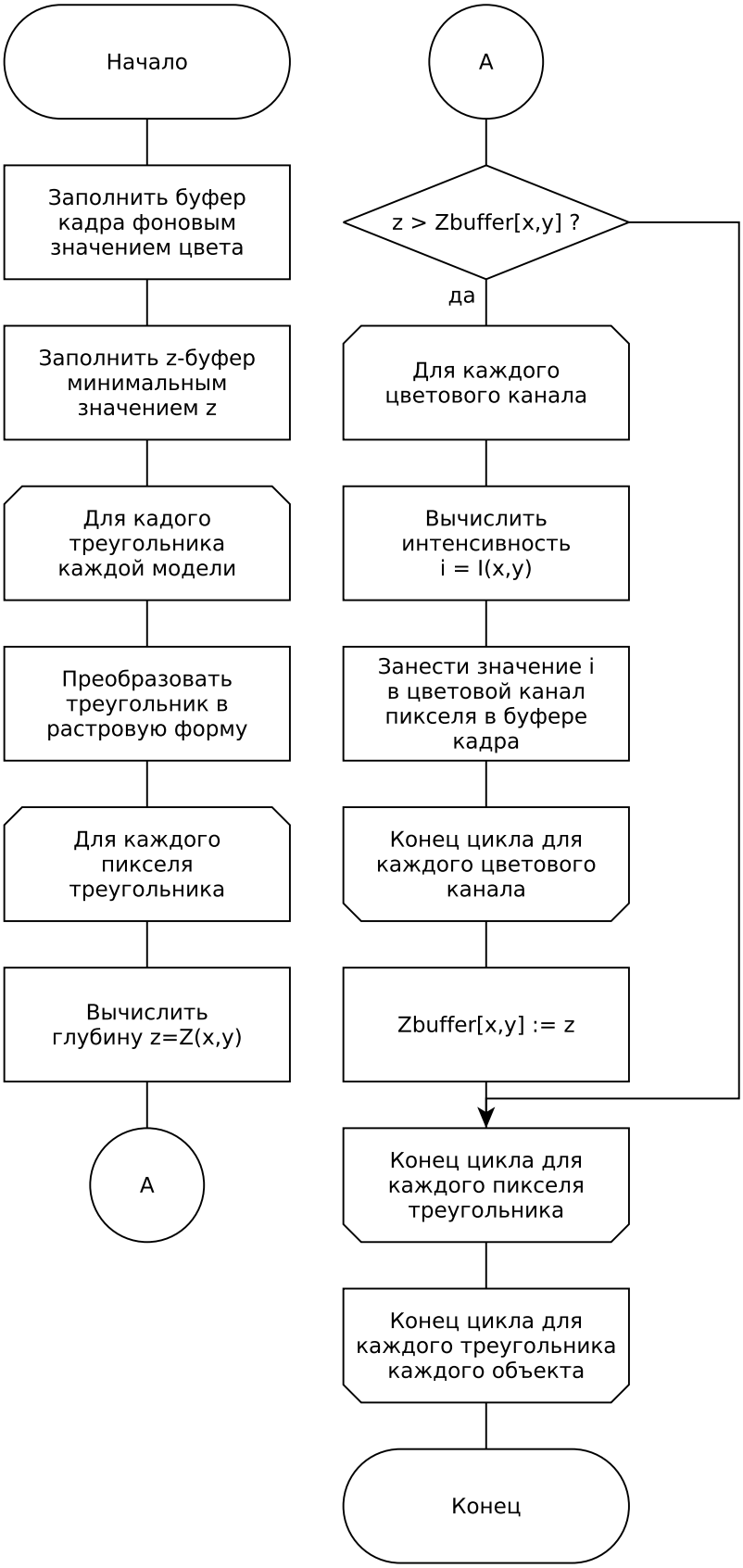

Рис. 2.2: Объединение алгоритма z-буфера и алгоритма закраски Гуро

Вывод

Были описаны структуры данных, способ моделирования поверхности, геометрические преобразования камеры, схема алгоритма синтеза изображения.

3 Технологический раздел

3.1 Выбор языка программирования и среды разработки

В качестве языка программирования для реализации данной курсовой работы был выбран высокопроизводительный язык C++ [8]. Этот язык поддерживает объектно-ориентированную модель разработки, что позволяет четко структурировать программу и легко модифицировать отдельные ее компоненты независимо от других. Язык C++ позволяет эффективно использовать ресурсы системы благодаря широкому набору функций и классов из стандартной библиотеки.

В качестве среды разработки был использован Qt Creator 4.10.2. Он обладает всем необходимым функционалом для написания, профилирования и отладки программ и создания графического пользовательского интерфейса. Данная среда поставляется вместе с фреймворком Qt 5.13.2 [9], классы и функции которого также были использованы при написании данного курсового проекта.

3.2 Структура программы

Структура программы представлена на рисунке 3.1 в виде схемы последовательных преобразований.

Рис. 3.1: Последовательность преобразований

3.2.1 Применение паттернов проектирования

Для структуры программы был выбран шаблон Model-View-Controller (MVC) [10], состоящий из трех отдельных компонентов: модели (model), которая предоставляет данные и реагирует на команды контроллера, изменяя своё состояние; представления (view), отвечающего за отображения данных модели; контроллера (controller), интерпретирующего действия пользователя, оповещая модель о необходимости внести изменения.

3.2.2 Схема классов

На рисунке 3.2 представлена схема классов.

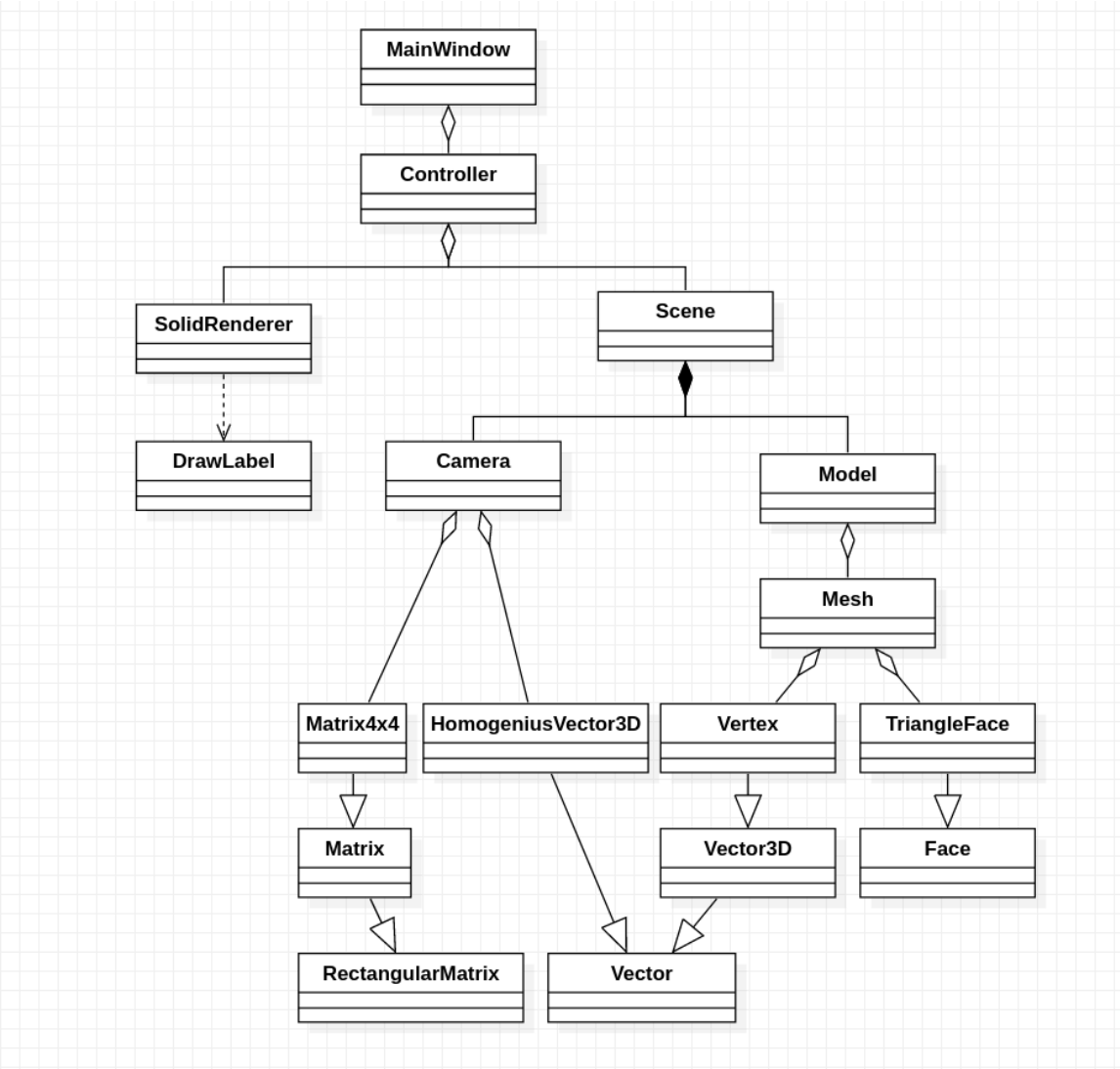

Рис. 3.2: Диаграмма классов программы

Классы `SolidRender` и `DrawLabel` входят в состав представления шаблона MVC, класс `Controller` — в состав контроллера, оставшиеся классы — в состав модели.

Разработанная программа состоит из следующих классов:

- Базовые математические классы:
  - `RectangularMatrix` — класс прямоугольных матриц;
  - `Matrix` — класс квадратных матриц;
  - `Matrix4x4` — класс матриц 4×4;
  - `Vector` — класс векторов;
  - `Vector3D` — класс векторов трёхмерных пространств;
  - `HomogeniusVector3D` — класс однородных векторов трёхмерных пространств;
- Классы для работы с моделями:
  - `Vertex` — класс вершин;
  - `Face` — класс примитивных поверхностей;
  - `TriangleFace` — класс треугольных примитивных поверхностей;
  - `Mesh` — Единица полигональной сетки.
- Классы сцены:
  - `Camera` — класс камеры с возможностью перемещения по сцене;
  - `Model` — класс аппроксимированной поверхности функции;
- Классы работы с изображением:
  - `Scene` — класс сцены, необходимой для визуализации;
  - `SolidRenderer` — класс для генерации изображения;
  - `Model` — класс для вывода изображения;
- Классы интерфейса:
  - `Controller` — класс контроллера;
  - `MainWindow` — класс главного окна сцены.

3.3 Интерфейс программы

Интерфейс программы был разработан с использованием редактора графического интерфейса Qt Designer. Главное окно программы можно условно разбить на две области: область настройки поверхностей и область изображения.

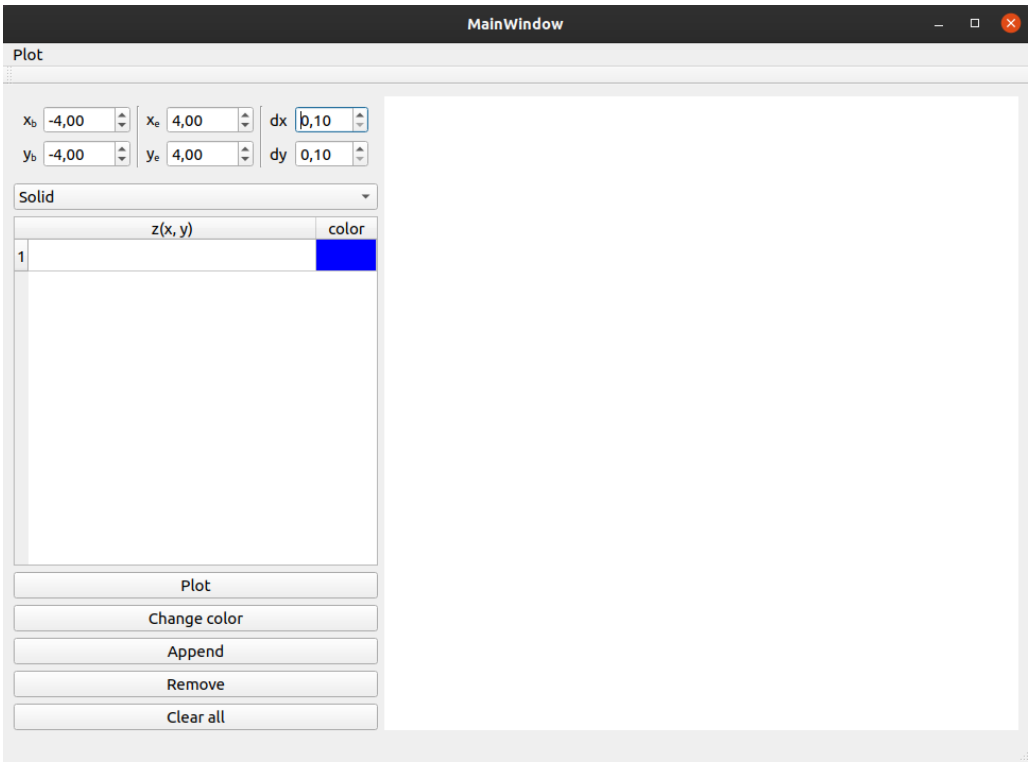

Рис. 3.3: Интерфейс программы при ее запуске

Для создания поверхности пользователю необходимо ввести функцию от двух переменных в список всех функций. Парсер математических выражений поддерживает все элементарные функции и распространённые константы. Чтобы добавить ещё одну функцию, необходимо нажать на кнопку `Append`. Для изменения цвета поверхности функции её необходимо выделить и, нажав на кнопку `Change color`, в диалоге выбора цвета выбрать наиболее подходящий. В списке можно выбрать произвольное количество функций. После нажатия на кнопку `Plot` поверхности всех выбранных функций появятся на экране. Используя мышь, можно перемещать камеру: колёсико мыши отвечает за приближение и отдаление картинки, левая кнопка — за вращение. С помощью кнопки `Remove` можно удалить выбранные функции, а нажатие на кнопку `Clear all` возвратит все настройки

программы к исходному виду. В левом верхнем виджете можно выбрать начальные и конечные значения x и y, а также их шаг.

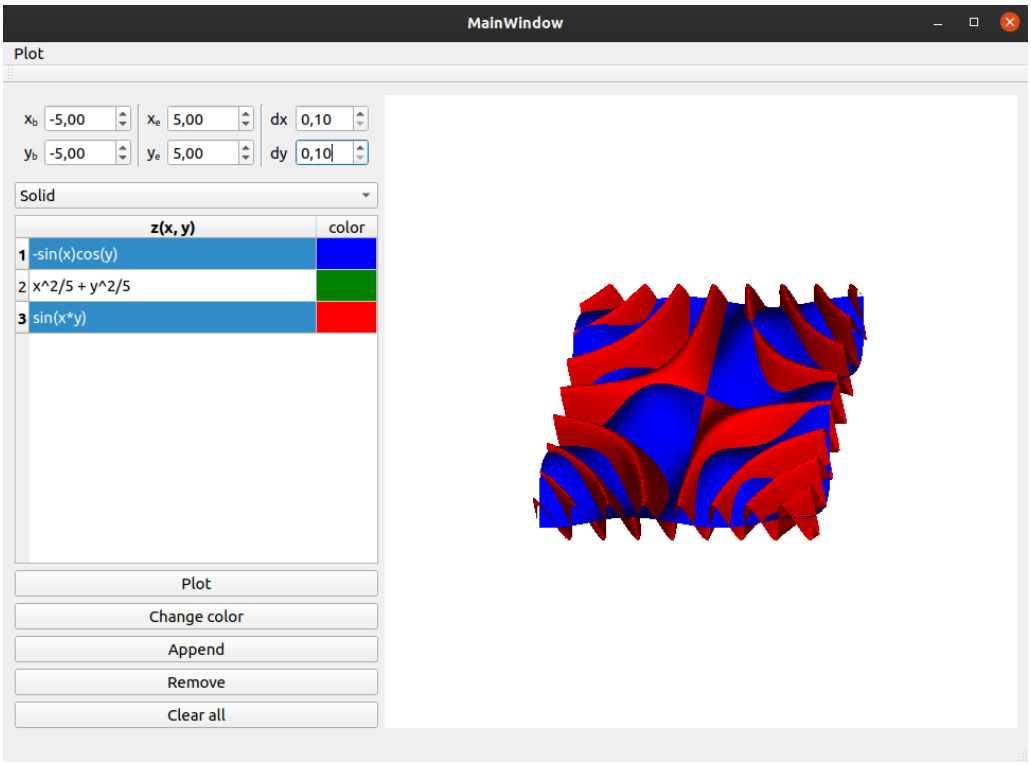

Рис. 3.4: Интерфейс программы при установленной скорости моделирования

3.4 Отладка и тестирование программы

Для отладки программы применялся отладчик gdb и библиотека QDebug, входящая в состав Qt, которая позволила выводить необходимую отладочную информацию в консоль во время выполнения программы. В качестве инструмента для анализа кода программы использовался статический анализатор ClangStatic Analyzer [11], который позволил на раннем этапе разработки обнаружить некоторые недочеты программного кода. Применяемый стандарт языка C++ не содержит средств для автоматической сборки мусора, то есть очистки динамически выделенной памяти, поэтому для контроля за динамически выделяемой памятью использовалась программа Valgrind [12].

Вывод

Правильный выбор инструментов разработки позволил эффективно реализовать алгоритмы, проводить отладку и тестирование программы, и выполнить технологический раздел курсовой работы.

4 Исследовательский раздел

4.1 Технические характеристики

Тестирование производительности программы производилось на компьютере со следующими техническими характеристиками:

- операционная система: Ubuntu 19.10 64-bit;
- память: 3,8 GiB;
- процессор: Intel® Core™ i3-6006U CPU @ 2.00GHz (4 логических ядра).

4.2 Зависимость производительности программы от количества полигонов каркасных моделей

Целью эксперимента является сравнение времени генерации изображения от количества полигонов, аппроксимирующих поверхности. Замеры производились для сцены 720×720 на одном потоке. В ходе эксперимента количество полигонов менялось от 100 до 1000 с шагом 100. Результаты представлены на рисунке 4.1.

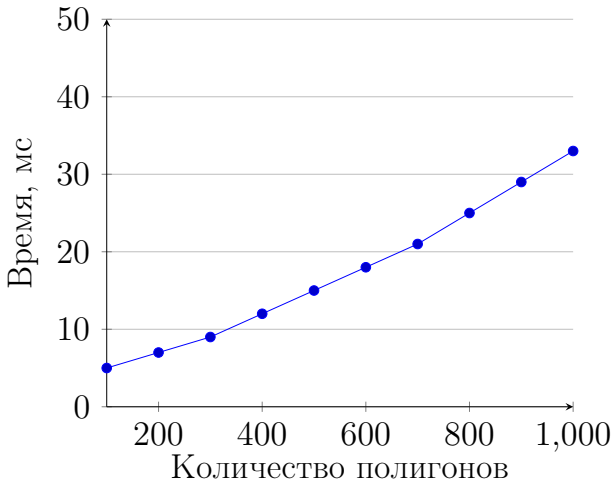

Рис. 4.1: Зависимость времени генерации изображения от количества полигонов, аппроксимирующих поверхности функций

4.3 Зависимость производительности программы от количества потоков

Целью эксперимента является сравнение времени генерации изображения от количества потоков. Замеры производились для сцены разрешением 600×600 пикселей с 500 полигонами. В ходе эксперимента количество потоков менялось от 1 до 64, на каждом шаге их число удваивалось. Результаты представлены на рисунке 4.2.

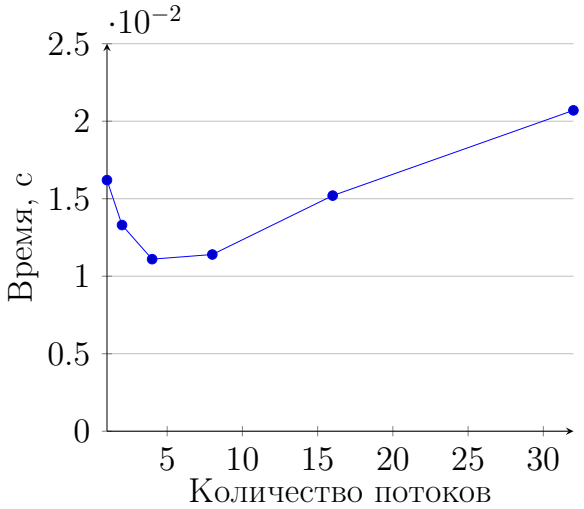

Рис. 4.2: Зависимость времени генерации изображения от количества потоков

Вывод

Как видно из результатов экспериментов, максимальной скорости работы удалось добиться при 4 потоках (быстрее, чем для 1 потока на 55%), что равно количеству логических ядер компьютера, на котором проводилось тестирование. Время генерации изображения линейно зависит от количество полигонов на сцене.

Заключение

В ходе проделанной работы, был выполнен сравнительный анализ алгоритмов удаления невидимых линий и поверхностей и алгоритмов закраски. Для разработки программы визуализации поверхностей, заданных функциями от двух переменных, была предложена оптимизация, заключающаяся в ускорении времени генерации изображения путём совмещения алгоритма Z-буфера и алгоритма закраски по Гуро.

Данный программный продукт может быть использован для демонстрации сложных графиков функций от двух переменных, их взаимного расположения. Предоставлены возможности настройки визуальных характеристик поверхностей, а так же изменение ориентации и положения камеры с помощью манипуляторов клавиатура и мышь.

В исследовательском разделе расчётно-пояснительной записки выяснено, что время генерации изображения линейно зависит от количества полигонов на сцене, а максимальной производительности можно добиться при количестве потоков, равном числу логических ядер процессора.

Благодаря использованию объектно-ориентированного подхода при разработке, программный продукт может быть легко модифицирован добавлением других типов моделей.

Программа курсовой работы полностью соответствует поставленному техническому заданию.

Литература

[1] Роджерс Д. Алгоритмические основы машинной графики. М: Мир, 1989. с. 501.

[2] Newman William M.; Sproull Robert F. Principle of interactive computer graphic. McGraw-Hill computer science series. McGraw-Hill College, 1978. с. 541.

[3] Данные об оборудовании пользователей Steam [Электронный ресурс]. URL: `https://store.steampowered.com/hwsurvey/` (Дата обращения: 12.10.2019).

[4] OpenGL FAQ/Depth Buffer [Электронный ресурс]. URL: `https://www.khronos.org/opengl/wiki/FAQ/Depth_Buffer` (Дата обращения: 25.10.2019).

[5] Depth Buffers (Direct3D 9) [Электронный ресурс]. URL: `https://docs.microsoft.com/en-us/windows/win32/direct3d9/depth-buffers` (Дата обращения: 25.10.2019).

[6] NVIDIA, Adobe to Bring Interactive, Photo-Real Ray Tracing to Millions of Graphic Designers [Электронный ресурс]. URL: `https://blogs.nvidia.com/blog/2018/10/15/adobe-max-interactive-photo-real-ray-tracing/` (Дата обращения: 25.10.2019).

[7] F. Dunn. 3D Math Primer for Graphics and Game Developmen. Taylor and Francis Group, LLC, 2011. с. 845.

[8] Working Draft, Standard for Programming Language C++. URL: `http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/n4713.pdf`. 2017.

[9] Qt Documentation [Электронный ресурс]. URL: `https://doc.qt.io/` (Дата обращения: 21.10.2019).

[10] Гамма Э. Приемы объектно-ориентированного проектирования. Паттерны проектирования. СПб.: Питер, 207. с. 366.

[11] Clang Static Analyzer Documentation [Электронный ресурс]. URL: https://clang.llvm.org/docs/ClangStaticAnalyzer.html (Дата обращения: 23.10.2019).

[12] Valgrind Documentation [Электронный ресурс]. URL: https://valgrind.org/docs/ (Дата обращения: 23.10.2019).