Содержание

Введение

В настоящее время компьютерная графика является неотъемлемой частью компьютерных технологий. Она позволяет наглядно выводить данные, создавать удобные и красивые интерфейсы. Также компьютерная графика используется в компьютерных играх, и в кино для создания эффектов и трёхмерных сцен.

Чтобы сделать возможным корректное преобразование цифровых данных в графические необходимо решить задачи синтеза сцены.

Зачастую существует несколько алгоритмов способных решить ту или иную задачу. Каждый алгоритм имеет свои достоинства и недостатки, одни чуть быстрее, другие потребляют меньше ресурсов и так далее. На программиста ложится задача выбора алгоритма необходимого для конкретной ситуации.

Цель данной работы построить сцену, на которой присутствует модель топлива и реализовать моделирование эффекта горения, включающее визуализацию пламени и симуляцию распространения огня по поверхности модели.

Задачи данной лабораторной работы:

- описать сцену и её элементы;
- проанализировать методы моделирования огня и выбрать метод, который будет использоваться в программе;
- проанализировать алгоритмы удаления невидимых граней и выбрать алгоритм, который будет использоваться в программе;
- проанализировать методы освещения и выбрать метод, который будет использоваться в программе;
- спроектировать и написать программное обеспечение;
- провести исследование написанного программного обеспечения.

1 Аналитическая часть

В данном разделе будут определены объекты сцены, рассмотрены и выбраны алгоритмы для синтеза сцены.

1.1 Формализация объектов сцены

Сцена состоит из следующих объектов:

- модель топлива, данный объект состоят из плоских выпуклых многоугольников, образующих многогранник и задаются в виде точек, являющихся вершинами многоугольника и рёбер их соединяющих, каждая модель имеет свои параметры горения, температуры самовозгорания и цвета;
- модель огня, данный объект состоит из точек генерации огня и частиц огня;
- камера, данный объект задаёт положение наблюдателя и вектор его взгляда;
- источник света, задаёт положение и интенсивность точечного источника света.

1.2 Методы генерации пламени

Существует два метода для генерации пламени:

- система частиц;
- уравнения горячих газов.

1.2.1 Система частиц

Реалистичность и проработанность огня будет зависеть от количества частиц и от законов их поведения, однако с увеличением количества частиц алгоритм соответственно начнёт работать медленнее. С помощью данного метода можно визуализировать огонь, используя маленькие спрайты. Пример использования системы частиц для моделирования огня изображён на рисунке 1.1.

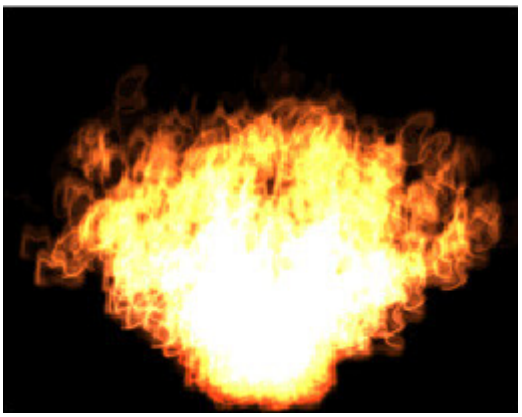

Рисунок 1.1 – Система частиц огня

1.2.2 Уравнения горячих газов

Данный метод полностью основан на физико-математическом подходе. В симуляции использовались несжатые уравнения Навье-Стокса для горячих газов, это позволило также смоделировать эффект расширения, вызванный испарением, и эффект текучести поднимающихся дыма и сажи. Однако данный подход сложно реализовать для работы в реальном времени, поскольку необходимо находить решение большого количества комплексных уравнений за время кадра. По этой причине данный способ не подходит.

Единственный подходящий метод, это система частиц, чтобы уменьшить их количество воспользуемся методом спрайтов, в таком случае на месте каждой частицы будет находится полупрозрачный спрайт из-за чего будет создаваться эффект пламени. Хотя данный подход и не даёт наиболее реалистичного пламени, он хорошо подходит для создания огня в режиме реального времени. Для реалистичного поведения пламени необходимо задать законы движения и взаимодействия его частиц. Для этого

необходимо учитывать следующее:

- тепловое расширения газа;
- закон Архимеда;
- сопротивление воздуха;
- тепловой обмен с окружающей средой [4].

1.3 Алгоритм распространения огня

Для распространения огня удобнее всего использовать систему точек генерации огня, производящую частички огня. Каждая точка такой системы имеет параметр температуры, который будет расти в зависимости от времени и материала модели. При достижении некоторой температуры вокруг точки генерации огня на некотором расстоянии будут возникать новые точки, если они не были созданы до этого и лежат на модели.

1.4 Анализ алгоритмов удаления невидимых линий и поверхностей

Для корректного отображение модели топлива необходимо устранить невидимые рёбра. При выборе алгоритма удаления невидимых рёбер нужно учитывать скорость работы алгоритма, так как при медленной работе алгоритма динамическая анимация огня соответственно замедлится и поведение огня станет нереалистичным. Рассмотрим несколько алгоритмов.

1.4.1 Алгоритм Робертса

Это метод основан на математическом подходе. Он работает в объектном пространстве, что является его плюсом. Алгоритм сперва удаляет рёбра, экранируемые самим телом. Затем каждое оставшееся ребро каждого тела сравнивается с каждым из оставшихся тел, таким образом определяется какие части рёбер экранируются. Так как происходит сравнение

каждого тела с каждым, то вычислительная трудоёмкость будет зависеть от квадрата количества тел [1]. Времязатратность алгоритма Робертса при большом количестве тел, является его главным минусом.

1.4.2 Алгоритм обратной трассировки лучей

Принцип обратной трассировки лучей состоит в том, что через каждую точку экрана как бы проводится обратный луч света до пересечения с ближайшим объектом сцены, далее из этой точки проводится луч в направлении источника света, таким образом, моделируется распространение света [1].

Плюсом данного алгоритма является то, что он выполняет множество задач, с его помощью возможен расчет теней, также можно моделировать многократные отражения и преломления за счёт этого достигается высокая степень реалистичности. Однако данный метод имеет множество недостатков, например, низкая скорость и высокая вычислительная стоимость расчетов из-за большого количества лучей для каждого из которых нужно вычислить пересечения. Также алгоритм не предусматривает учёт вторичного освещения от диффузно отраженного объектами света. Возникают резкие границы самих объектов и цветовых переходов тени, подсветки, прозрачности.

1.4.3 Алгоритм Варнока

В отличие от алгоритма Робертса, алгоритм Варнока работает не в объектном пространстве, а в пространстве образа. Его главная идея заключается в том, чтобы на обработку тех областей, которые содержат мало информации тратилось мало времени и вычислений, а на области с высоким информационным содержимым тратилась большая вычислительная мощность. 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. Конкретная реализация алгоритма Варнока зависит от метода разбиения

окна и от деталей критерия, используемого для того, чтобы решить, является ли содержимое окна достаточно простым. внешним, если он целиком находится вне окна [1].

Достоинством алгоритма Варнока можно назвать сравнительно низкую времязатратность, зависящую от эффективности разбиений количества многоугольников в кадре и необходимого разрешения. Однако из-за произвольной разбивки, выполняемой для распознавания, в изображениях возникают дефекты.

1.4.4 Алгоритм Вейлера-Азертона

Данный алгоритм является оптимизацией алгоритма Варнока за счёт сокращения выполняемых разбиений, перейдя от прямоугольных разбиений к разбиениям вдоль границ многоугольников.

Для оптимизации в алгоритм были добавлены пункты:

- предварительная сортировка по глубине;
- отсечение по границе ближайшего к точке наблюдения многоугольника, называемое сортировкой многоугольников на плоскости;
- удаление многоугольников, экранируемых более близкими к точке наблюдения многоугольниками.

Эффективность такого метода, как и алгоритм Варнока, зависит от эффективности разбиений и точно также, как и в алгоритме Варнока при построении изображения возникают дефекты [2].

1.4.5 Метод Z-буфера

Этот алгоритм работает в пространстве изображения. Главным преимуществом алгоритма является его простота. Кроме того, этот алгоритм делает тривиальной визуализацию пересечений сложных поверхностей. Сцены могут быть любой сложности. Поскольку габариты пространства изображения фиксированы, оценка вычислительной трудоемкости алгоритма не более чем линейна. Элементы сцены не нужно предварительно сортировать по приоритету глубины. За счёт этого экономится вычислительное

время, затрачиваемое на сортировку по глубине [1]. Недостатком алгоритма является большой объем требуемой памяти, однако при современных технологиях это не является проблемой, а также можно использовать z-буфер с сканирующей строкой, что уменьшит необходимую память. Другой недостаток алгоритма состоит в трудоемкости реализации эффектов, связанных с полупрозрачностью, и ряда других специальных задач, повышающих реалистичность изображения, однако в моей работе это и не требуется.

Хотя метод обратной трассировки лучей и является наиболее подходящим так как обеспечивает одновременно удаление невидимых поверхностей и создание реалистичного света, его времязатратность не даёт возможности воспроизводить анимацию огня в реальном времени без больших вычислительных мощностей. Наилучшим образом тут подойдёт алгоритм Z- буфера. Данный алгоритм будет самым быстро действенным из всех перечисленных, простым в реализации и создающим малое количество дефектов изображения.

1.5 Перевод точки в экранные координаты

Для отображения модели на экране необходимо перевести координаты её точек из объектных в экранные. Предполагается, что камера находится в нуле координат и вектор взгляда совпадает с осью z. В таком случае нам необходимо в начале применить к точкам преобразования для перевода их в пространство камеры. После того как все преобразования сделаны необходимо отмасштабировать x и y пропорционально их удалённости по оси z и в зависимости от угла обзора камеры, это демонстрируется на рисунках 1.2, 1.3;

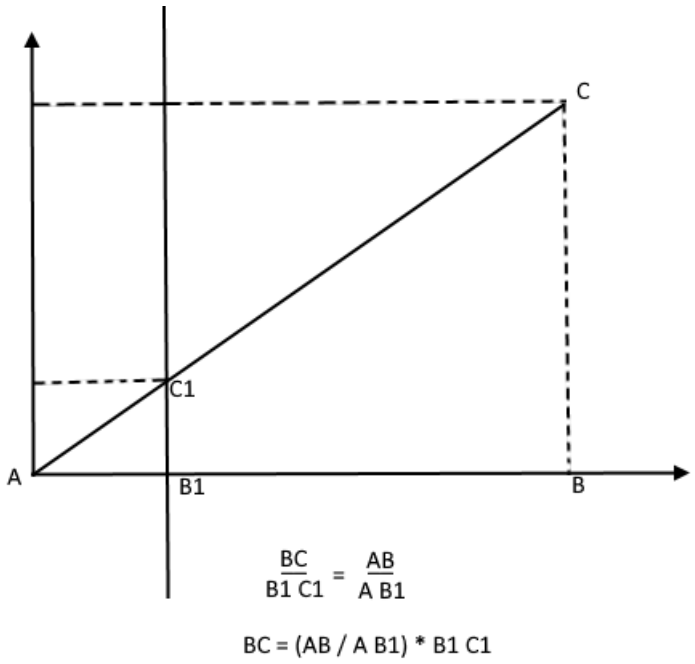

Рисунок 1.2 – Геометрия преобразований 1

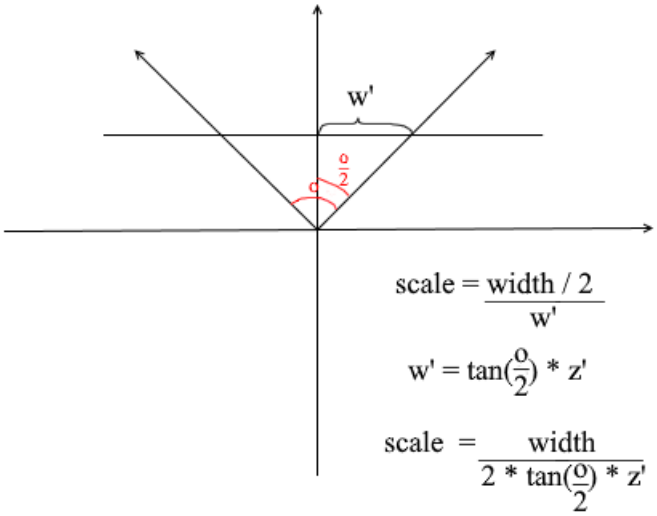

Рисунок 1.3 – Геометрия преобразований 2

Получим формулы 1.1, 1.2 преобразования координат x и y.

$$x' = x/(z+1) * scale \tag{1.1}$$

$$y' = y/(z+1) * scale \tag{1.2}$$

1.6 Преобразования

Для изменения сцены необходимо реализовать простейшие преобразования над объектами, такие как сдвиг, и поворот. Для этих целей будут использоваться формулы 1.6, 1.5, 1.4, 1.3.

Сдвиг:

$$\begin{cases} X = X + Dx, \\ Y = Y + Dy, \\ Z = Z + Dz. \end{cases} \tag{1.3}$$

Поворот вокруг:

OX:

$$\begin{cases} z' = -y * sin(ax) + z * cos(ax), \\ x' = x, \\ y' = y * cos(ax) + z * sin(ax). \end{cases} \tag{1.4}$$

OY:

$$\begin{cases} z' = -x * sin(ay) + z * cos(ay), \\ x' = x * cos(ay) + z * sin(ay), \\ y' = y. \end{cases} \tag{1.5}$$

OZ:

$$\begin{cases} z' = z, \\ x' = x * cos(az) - y * sin(az), \\ y' = x * sin(az) + y * cos(az). \end{cases} \tag{1.6}$$

1.7 Освещение

Для того чтобы объекты сцены было видно, и они выглядели объёмно, на сцене должно присутствовать освещение. Существует три компоненты освещения:

- фоновое;
- рассеяное;
- зеркальное.

Фоновое освещение. Данный вид освещения используется для того чтобы сцена не была тёмной. За счёт данного освещения объекты не будут абсолютно чёрными и будут видны его очертания. Чтобы имитировать это, используем некоторую константу освещения, которая всегда будет придавать объекту некоторый оттенок, данная константа должна быть сравнительно небольшой, чтобы не сильно перебивать другие виды освещений.

$$Ia = ka * ia, \tag{1.7}$$

где

Ia – фоновая составляющая освещенности в точке;

ka – свойство материала воспринимать фоновое освещение;

ia – мощность фонового освещения.

Диффузное освещение имитирует воздействие на объект направленного источника света. Это наиболее визуально значимый компонент модели освещения. Чем большая часть поверхности объекта обращена к источнику света, тем ярче он будет освещен.

$$I_d = K\_d * i\_d * (\vec{L} * \vec{N}), \tag{1.8}$$

где

Id– рассеянная составляющая освещенности в точке;

kd– свойство материала воспринимать рассеянное освещение;

id– мощность рассеянного освещения;

L – направление из точки на источник;

N - вектор нормали в точке.

Освещение зеркальных бликов - это яркие пятна света или блик. По цвету зеркальные блики часто ближе к цвету источника света, чем к цвету объекта. Яркость и размер бликов зависят от того, насколько гладкая поверхность [2]. Поверхности объектов будут однотонными, следовательно, будут иметь свои показатели поглощения и отражения цветов. Формула для вычисления зеркальных бликов выглядит следующим образом:

$$I_s = K\_s * i\_s * [(\vec{V} * \vec{R})]^a, \tag{1.9}$$

где

Is – зеркальная составляющая освещенности в точке;

ks – коэффициент зеркального отражения;

is – мощность зеркального освещения;

R – направление отраженного луча;

V - направление на наблюдателя;

a - коэффициент блеска, свойство материала;

1.7.1 Метод Фонга и метод Гуро.

Для вычисления зеркалного и рассеянного света существует два подхода:

- метод Гуро обеспечивает непрерывность освещенности за счет ее билинейной интерполяции: после определения значений освещенности в вершинах полигона применяют линейную интерполяцию вдоль сторон полигона, а потом – линейную интерполяцию между сторонами полигона вдоль каждой из сканирующих строк, пересекающих полигон (при этом освещенность рассчитывается для каждого пикселя соответствующего интервала сканирующей строки);

- метод Фонга аналогичен методу Гуро, но при его использовании для определения цвета в каждой точке интерполируются не интенсивности отраженного света, а векторы нормалей.

Метод Фонга требует больше вычислений, чем метод Гуро, однако даёт гораздо лучшее качество изображения рисунок 1.4.

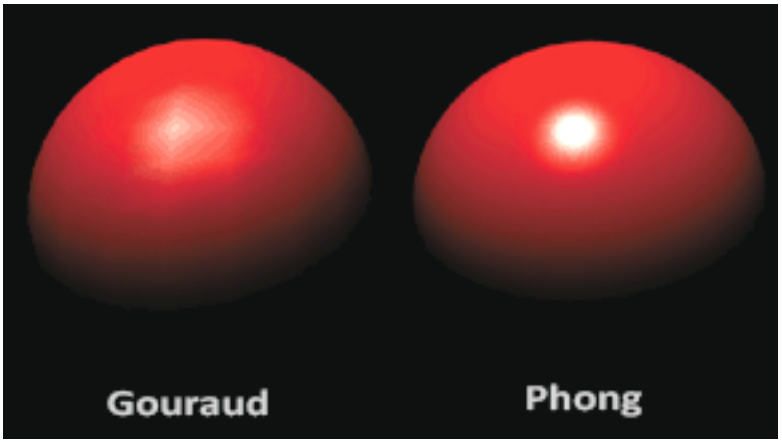

Рисунок 1.4 – Закраска по Фонгу и Гуро

1.7.2 Простая закраска

Данный вид закраски подразумевает только вычисления цвета по формуле без необходимости интерполяции интенсивности или вектора нормали. Поскольку в данном случае используется модель с плоскими гранями, то интерполяция цвета не требуется, поэтому будем применять простую закраску.

Общая формула 1.10 для закраски будет суммой формул 1.9,1.8, 1.7 поделённой на квадрат расстояния до источника света:

$$I = kaia + (K\_d * i\_d * (\vec{L} * \vec{N}) + K\_s * i\_s * ([\vec{V} * \vec{R}]^a)/(d^2) \qquad (1.10)$$

Интерполяция вектора нормали производится не будет.

Вывод из аналитической части

В данном разделе были рассмотрены алгоритмы удаления невидимых линий и поверхностей, методы освещения и моделирования огня. В качестве алгоритма удаления невидимых линий был выбран Z-буфер, для моделирования огня была выбрана система частиц с наложением на них спрайтов, в качестве метода затенения был выбран простой алгоритм.

2 | Конструкторская часть

В данном разделе будут рассмотрены требования к программе и алгоритмы, используемые в программе.

2.1 Требования к программе

Программа должна предоставлять следующие возможности:

- визуальное отображение сцены;
- добавление нового очага возгорания в сцену;
- выбор материала модели из набора (дерево, древесный уголь, каменный уголь, торф);
- задать положение и интенсивность точечного источника света;
- перемещение камеры по сцене и поворот камеры по вертикали и горизонтали.

2.2 Схема взаимодействия классов

На рисунках 2.1, 2.2, 2.3 приведены UML диаграммы взаимодействия классов.

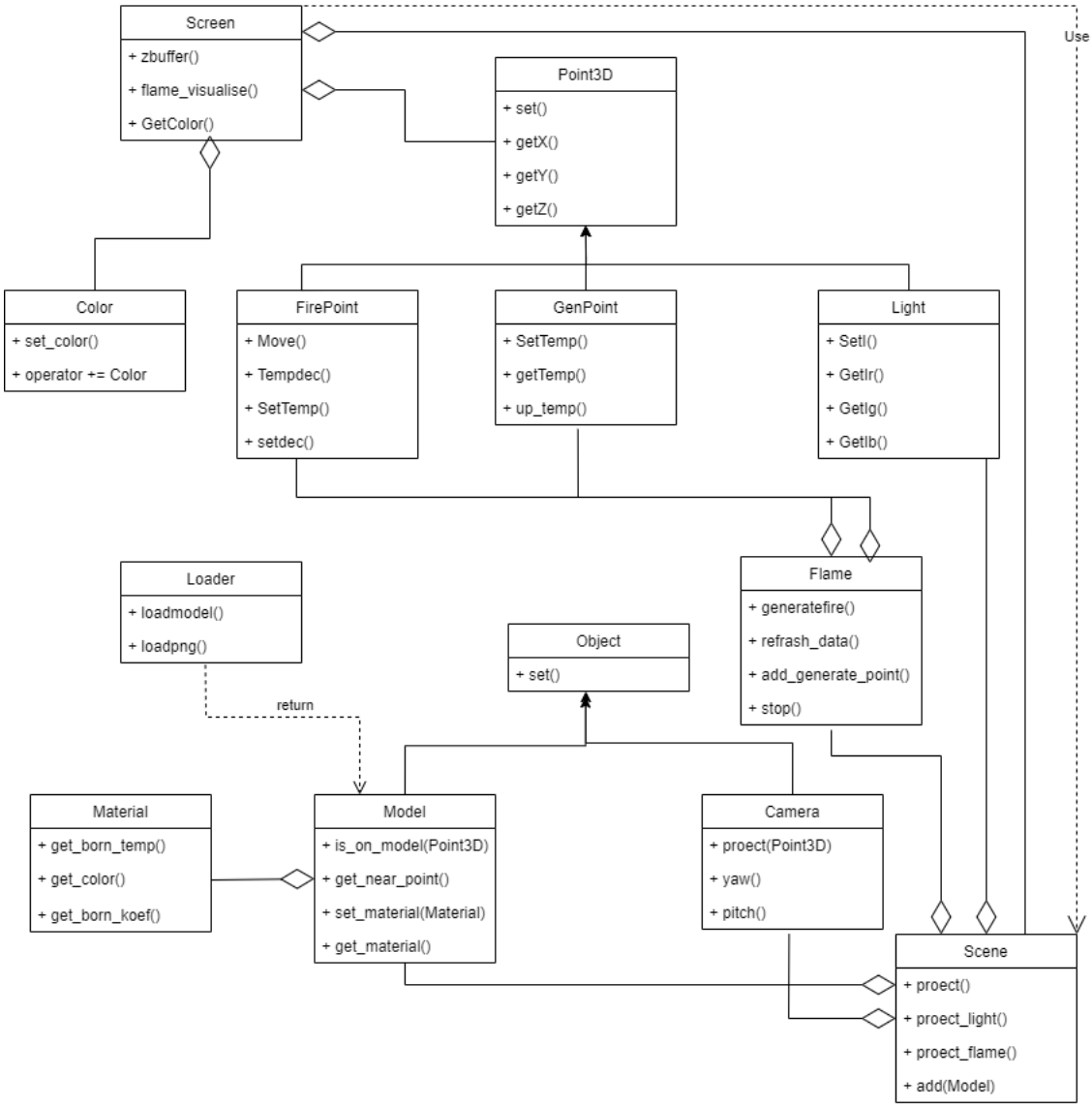

Рисунок 2.1 – UML Диаграма часть 1

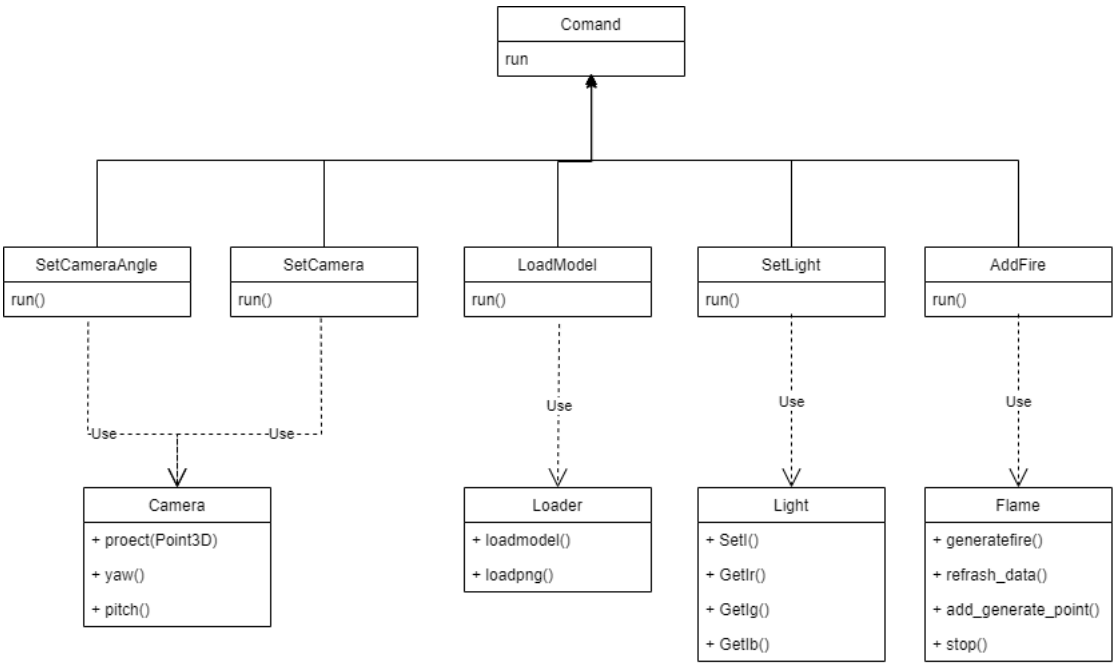

Рисунок 2.2 – UML Диаграма часть 2

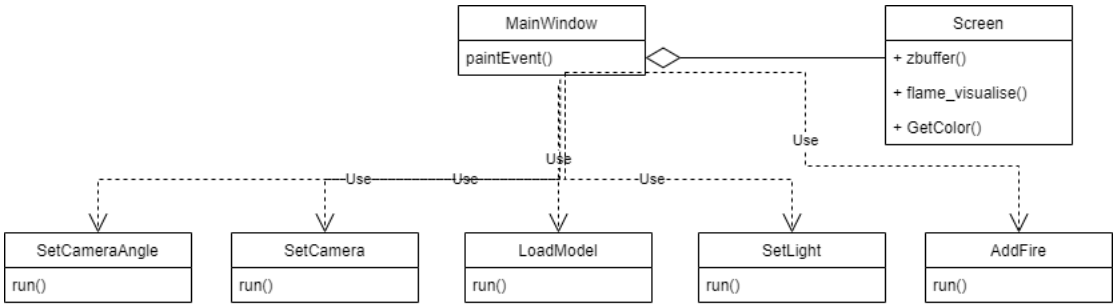

Рисунок 2.3 – UML Диаграма часть 3

2.3 Описание алгоритмов

Далее приведены схемы алгоритмов для создания программного обеспечения.

2.3.1 Основной алгоритм

На рисунке 2.4 приведена схема основного алгоритма.

Рисунок 2.4 – Схема основного алгоритма

2.3.2 Алгоритм Z-буффера

На рисунках 2.5, 2.6 приведена схема алгоритма Z-буффера.

Рисунок 2.5 – Схема алгоритма Z-буфера часть 1

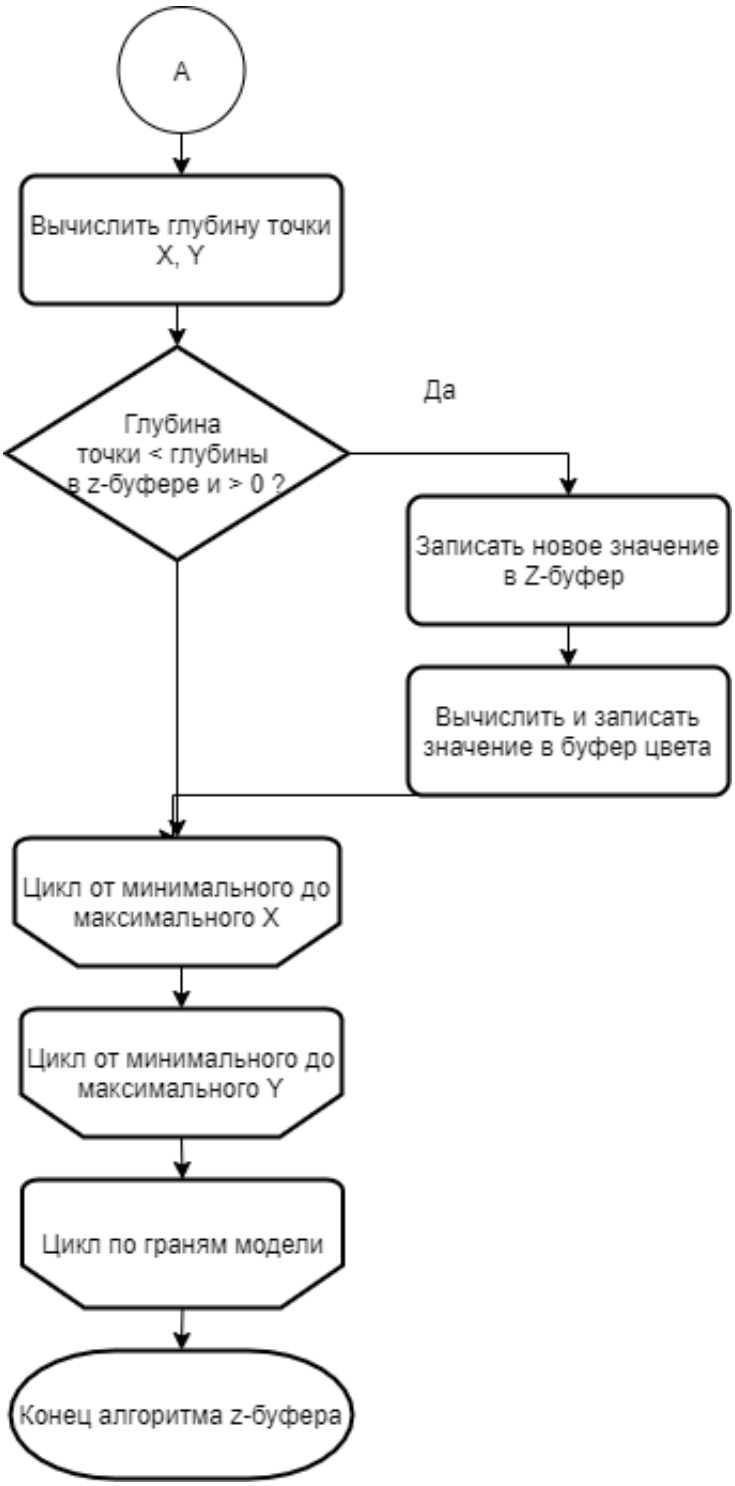

Рисунок 2.6 – Схема алгоритма Z-буфера часть 2

2.3.3 Алгоритм моделирования огня

На рисунке 2.7 приведена схема алгоритма моделирования огня.

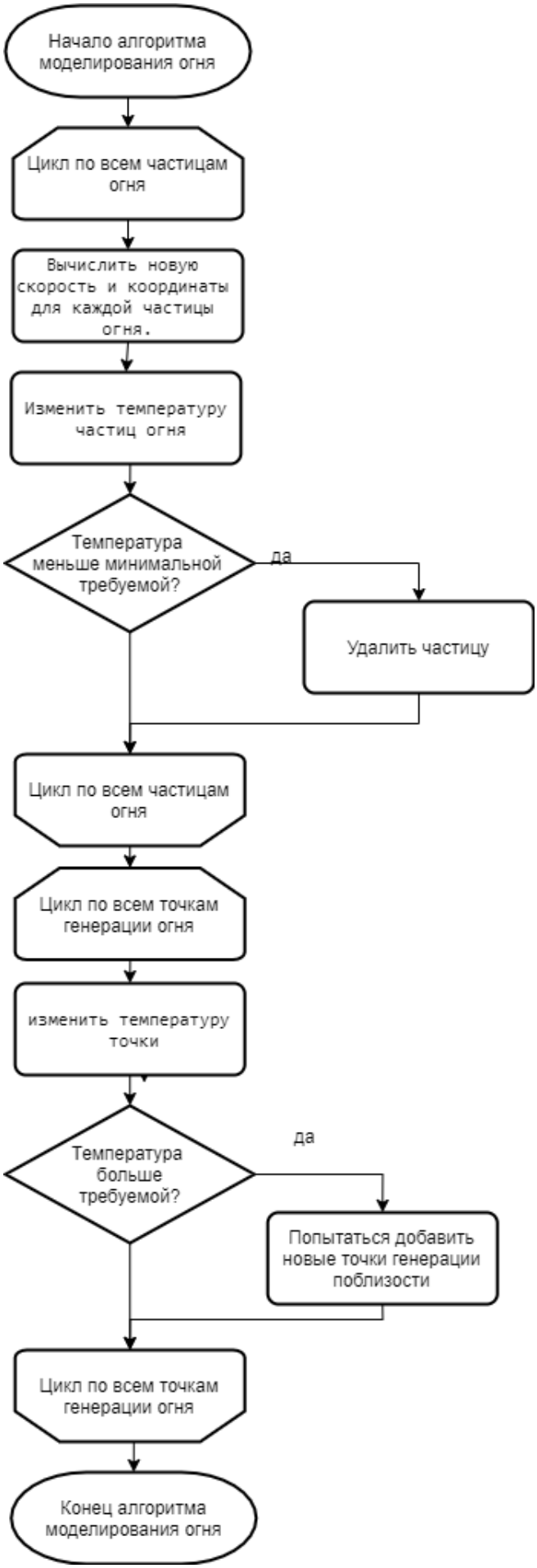

Рисунок 2.7 – Схема алгоритма моделирования огня

2.3.4 Алгоритм визуализации огня

На рисунке 2.8 приведена схема алгоритма визуализации огня.

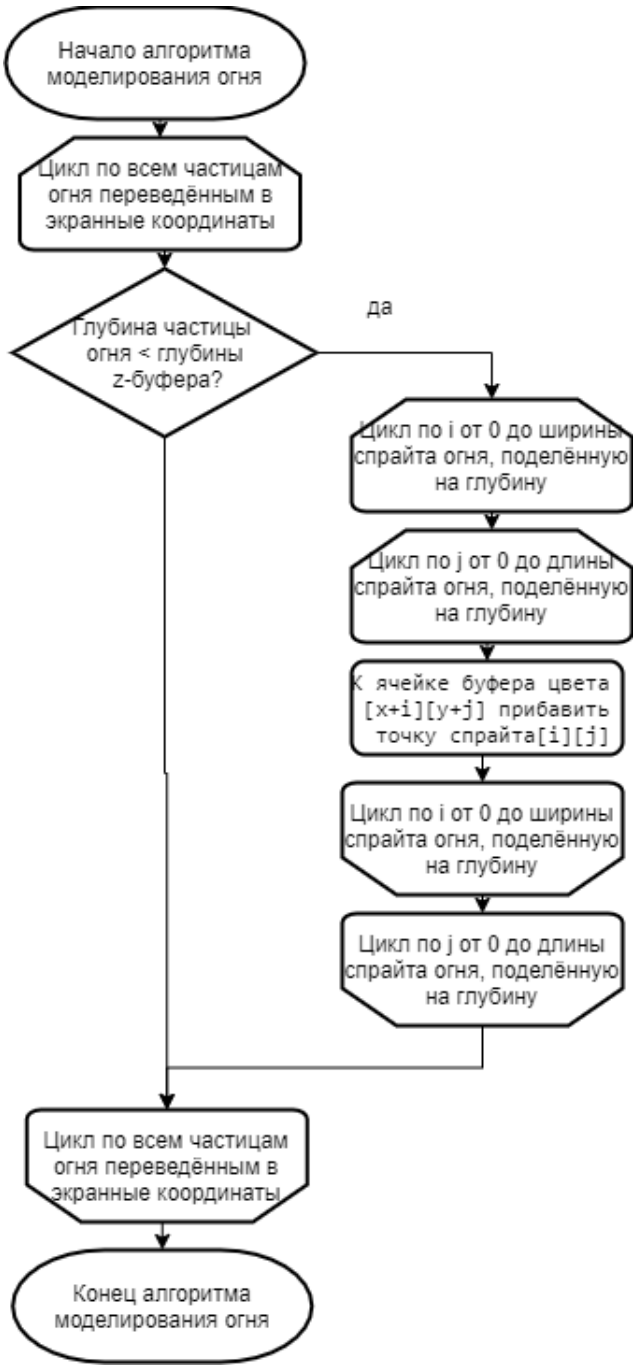

Рисунок 2.8 – Схема алгоритма визуализации огня

2.4 Распараллеливание задач

Если сделать последовательным выполнение вычислений и отрисовки, это приведёт к зависанию приложения так как основной поток будет

занят циклом вычислений и не сможет откликаться на команды пользователя. Поэтому для выполнения вычислений создаётся отдельный поток, который при окончании вычислений будет вызывать событие отрисовки в главном потоке. В таком случае для корректной отрисовки буфера цвета необходимо, чтобы поток выполняющий вычисления дожидался окончания отрисовки. Для этого будет использоваться мьютекс.

2.5 Модель горючего материала

Для демонстрации работы программы была выбрана модель кубика с вершинами в точках , представленных в таблице 2.1, данный выбор обусловлен удобством данной модели для установки точки горения и наблюдения за распространением огня. Пользователь может задавать тип материала модели, однако форма и размеры модели будут оставаться неизменными.

Таблица 2.1 – Координаты точек куба

№ точки	координата x	координата y	координата z
1	1	1	1
2	-1	1	1
3	1	-1	1
4	-1	-1	1
5	1	1	-1
6	-1	1	-1
7	1	-1	-1
8	-1	-1	-1

Вывод из конструкторской части

В данном разделе были приведены алгоритмы для создания программного обеспечения, рассмотрены схемы взаимодействия классов, приведены схемы алгоритмов. Был сделан вывод о необходимости распараллеливания задач.

3 | Технологическая часть

В данном разделе рассматриваются, производится выбор средств реализации, рассматривается строение программного обеспечения.

3.1 Средства реализации

В качестве языка программирования был выбран C++ по следующим причинам:

- имеется опыт программирования на это языке, что сократит время написания и отладки программы;
- данный язык предоставляет возможность объектно-ориентированного программирования, что позволяет рассматривать элементы сцены как объекты, это сделает взаимодействие между элементами сцены более понятными и удобными для разработчика;
- для данного языка имеется множество библиотек.

В качестве среды разработки была выбрана «QT Creator» по следующим причинам:

- данная среда разработки бесплатна в пользовании студентами;
- она имеет множество удобств, которые облегчают процесс написания и отладки кода;
- имеется опыт работы в данной средой разработки, что сократит время изучения возможностей среды.

3.2 Структура приложения

Приложение состоит из следующих модулей:

- INCLUDS.h - файл подключения основных библиотек и определений;

- Objects.h - файл определения классов;
- Primitiv.h - файл определения классов;
- algoritms.h - файл определения классов;
- comands.h - файл определения классов;
- loader.h - файл определения классов;
- mainwindow.h - файл определения классов;
- mutexs.h - файл определения классов;
- screen.h - файл определения классов;
- mainwindow.ui - файл содержащий дизайн интерфейса;
- alghoritms.cpp - файл функций классов;
- comands.cpp - файл функций классов;
- loader.cpp - файл функций классов;
- main.cpp - файл содержавший функцию main();
- mainwindow.cpp - файл функций классов;
- object.cpp - файл функций классов;
- primitiv.cpp - файл функций классов;
- screen.cpp - файл функций классов.

Также в директории с исполняемым файлом находятся следующие файлы:

- coub.obj - Файл содержащий описание модели в формате obj;
- fire0.bmp - 24 битный bmp файл содержащий спрайт частиц огня.

3.3 Интерфейс приложения

На рисунке 3.1 изображён интерфейс приложения.

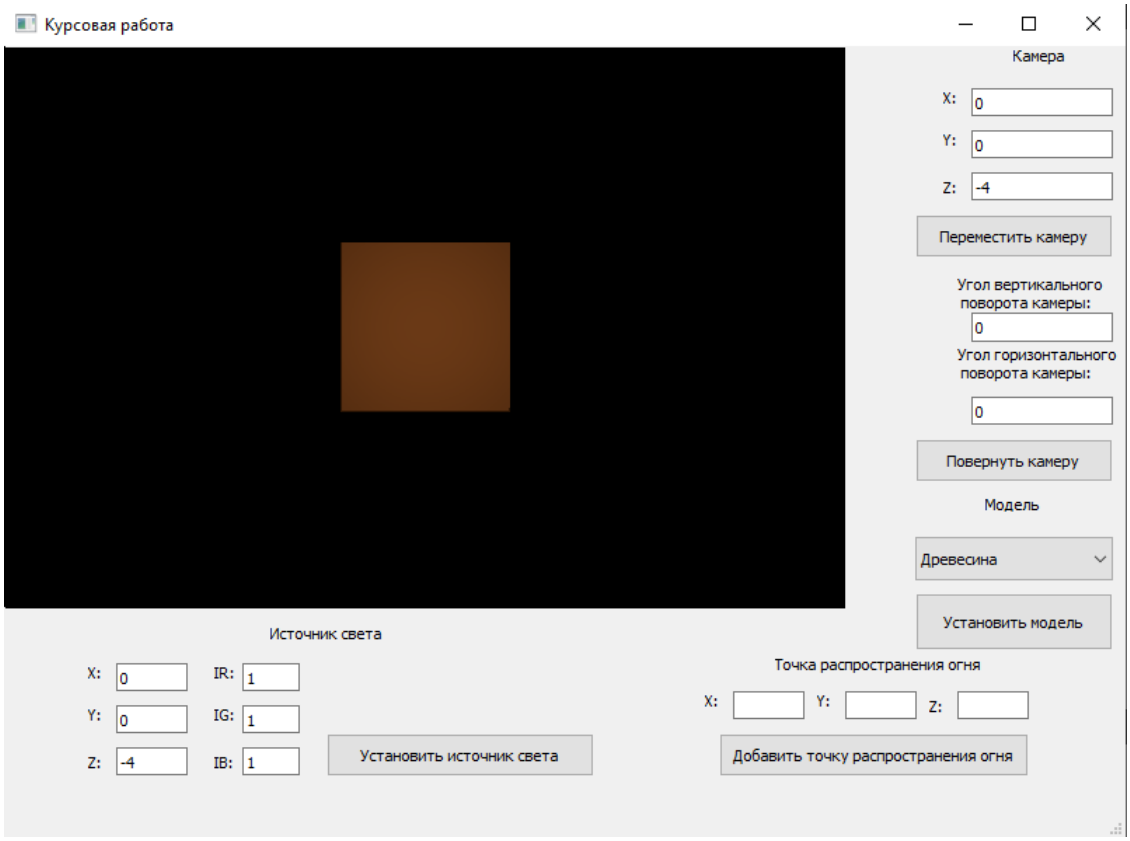

Рисунок 3.1 – Интерфейс приложения

Интерфейс поделён на 4 логические части расположенных на форме:

- камера;
  - поле "X"задаёт координату x камеры в виде числа с десятичной дробью;
  - поле "Y"задаёт координату y камеры в виде числа с десятичной дробью;
  - поле "Z"задаёт координату z камеры в виде числа с десятичной дробью;
  - поле "Угол вертикального поворота камеры"задаёт вертикальный наклон камеры в градусах в виде числа с десятичной дробью;
  - поле "Угол горизонтального поворота камеры"задаёт поворот камеры по оси OY в градусах в виде числа с десятичной дробью;

- кнопка "Переместить камеру при корректном заполнении полей "X "Y "Z п
  реносит камеру в координаты, введённые в полях "X "Y "Z";
- кнопка "Повернуть камеру при корректном заполнении полей "Угол вертикального поворота камеры "Угол горизонтального поворота камеры задаёт углы поворота камеры, введённые в полях "Угол вертикального поворота камеры "Угол горизонтального поворота камеры";

- модель;
  - в окошке с выбором можно выбрать тип материала модели из набора (дерево, каменный уголь, древесный уголь, торф);
  - кнопка "Установить модель"меняет материал модели на выбранный, при этом все точки распространения огня, присутствующие на модели удаляются;

- источник света;
  - поле "X"задаёт координату x источника света в виде числа с десятичной дробью;
  - поле "Y"задаёт координату y источника света в виде числа с десятичной дробью;
  - поле "Z"задаёт координату z источника света в виде числа с десятичной дробью;
  - поле "IR"задаёт интенсивность красной составляющей цвета источника света в виде целого числа;
  - поле "IG"задаёт интенсивность зелёной составляющей цвета источника света в виде целого числа;
  - поле "IB"задаёт интенсивность синей составляющей цвета источника света в виде целого числа;
  - кнопка "Установить источник света при корректном заполнении полей "X "Y "Z "IR "IG "IB переносит источник света в координаты введённые в полях "X "Y "Z и задаёт его интенсивность, введённую в полях "IR "IG "IB";

- точка распространения огня;
  - поле "X"задаёт координату x точки распространения в виде числа с десятичной дробью;
  - поле "Y"задаёт координату y точки распространения в виде числа с десятичной дробью;
  - поле "Z"задаёт координату z точки распространения в виде числа с десятичной дробью;
  - кнопка "Добавить точку распространения огня при корректном заполнении полей "X "Y "Z"и при условии, что тока с соответствующими координатами принадлежит поверхности модели.

Начиная от верхнего левого угла на форме расположен экран размером 600 X 400 на который выводится синтезированное изображение.

3.4 Листинги алгортмов программы

В листингах 3.1 - 3.6 приведены реализации основных алгоритмов, используемых в программе.

Листинг 3.1 – Функция Z-буфера часть 1

```
void Screen::zbuffer(std::vector <Model> &m)
{
    std::vector <List> edges = m[0].edges;
    int addnum = -1;
    int ystart, yend, xfirst, xlast, buf, intersect;
    std::vector <int> intersections;
    Point3D K, N;
    double CalcZ;
    fill(Zbuf, Cbuf);
    for(int l = 0; l < edges.size(); l++)
    {
    K = CalcKoef(edges[l], P, addnum);
    N = Normal(edges[l], P, addnum);
    MinMaxY(edges[l], P, ystart, yend, addnum);
    for (int y = ystart; y <= yend; y++)
    {
        edges[l].to_end();
        buf = edges[l].get_npoint() + addnum;
        for (edges[l].to_begin();!edges[l].is_end(); edges[l].next
           ())
        {
            intersect = intersection(P[buf].getX(),P[buf].getY(),P
               [edges[l].get_npoint()+addnum].getX(),P[edges[l].
               get_npoint()+addnum].getY(), y);
            buf = edges[l].get_npoint() + addnum;
            if (intersect != -1)
                intersections.push_back(intersect);
        }
```

Листинг 3.2 – Функция Z-буфера часть 2

```
        intersect = intersection(P[buf].getX(),P[buf].getY(),P[
            edges[l].get_npoint()+addnum].getX(),P[edges[l].
            get_npoint()+addnum].getY(), y);
        if (intersect != -1)
            intersections.push_back(intersect);
        if (intersections.size() < 2)
            continue;
        MinMaxX(intersections, xfirst, xlast);
        intersections.clear();
        for (int x = xfirst; x <= xlast; x++)
        {
            CalcZ = x*K.getX() + y*K.getY() + K.getZ();
            if (Zbuf[x][y] > CalcZ and CalcZ > 0)
            {
                Zbuf[x][y] = CalcZ;
                Point3D P0((x - WINWIDTHD2) * ((CalcZ + 1 > EPS) ?
                    (CalcZ + 1) : 0.001) / MKOEF, (y - WINHEIGHTD2
                    ) * ((CalcZ + 1 > EPS) ? (CalcZ + 1) : 0.001) /
                    MKOEF, CalcZ);
                Cbuf[x][y] = pixelcolor(m[0].mat.get_color(), P0,
                    N, Lights);
            }
        }
    }
    }
}
```

Листинг 3.3 – Функция визуализации огня

```
void Screen::flame_visualise()
{
    double x, y, z;
    double e;
    for (Point3D P : Flame)
    {
        x = P.getX();
        y = P.getY();
        z = P.getZ();
        double K = z*FKOEF;
        int maxx = round(flame_mask.size()/K);
        int maxy;
        if (maxx > 0)
            maxy = round(flame_mask[0].size()/K);
        int maxxd2 = maxx / 2;
        if (x > -maxx and x < WINWIDTH + maxx and y > -maxy and y
          < WINHEIGHT + maxy and abs(z) > 0.5)
            for(int i = 0; i < maxx and (x+i-maxxd2)>0 and (x+i-
               maxxd2)<WINWIDTH; i++)
            {
                int maxyd2 = maxy / 2;
                for(int j = 0; j < maxy and (y+j-maxyd2)>0 and (y+j
                   -maxyd2)<WINWIDTH; j++)
                {
                    if (Zbuf[x+i-maxxd2][y+j-maxyd2] >= z)
                        Cbuf[x+i-maxxd2][y+j-maxyd2] += flame_mask
                           [i*K][j*K];
                }
            }
    }
}
```

Листинг 3.4 – Функция обновления данных частиц огня

```
void Flame::refrash_data()
{
    if (fire.size() == 0)
        return;
    int envtemp = MinTemp;
    long long sr = 0;
    int ch = 0;
    double dist = 0;
    std::list <FirePoint> :: iterator it = fire.begin();
    for (;it!=fire.end();it++)
    {
        sr = 0;
        ch = 0;
        for (FirePoint &F: fire)
        {
            dist = F.sqrdistP(*it);
            if (dist > EPS and dist < 0.01)
            {
                sr += F.GetT();
                ch++;
                if (ch == 10)
                    break;
            }
        }
        for (int i = ch; i < 10; i++)
        {
            sr += MinTemp;
            ch++;
        }
        it->setdec(sr/ch);
        it->Tempdec();
        if (it->GetT() < MinTemp or it->get_speed() < EPS)
            fire.erase(it);
        else
            it->Move();
    }
    for (int i = 0; i < (5 - (fire.size()/generate.size())) ; i++)
        generatefire();
};
```

Листинг 3.5 – Цикл работы

```
void actionThread(thr data)
{
    while(true)
    {
     comand_mtx.lock();
     data.scene.proect(data.Points);
     data.scene.proect_light(data.light);
     data.scene.proect_flame(data.FlameP);
     data.flame.refrash_data();
     data.flame.check_genpoints(data.model[0]);
     comand_mtx.unlock();
     draw_mtx.lock();
     data.screen.zbuffer(data.model);
     data.screen.flame_visualise();
     draw_mtx.unlock();
     data.m.update();
    }

}
```

Листинг 3.6 – Функция перевода точки в экранный координаты

```
Point3D Camera::proect(Point3D &P)
{
    double sx = P.getX() - position.getX();
    double sy = P.getY() - position.getY();
    double sz = P.getZ() - position.getZ();
    double x = sx * cosY + sz * sinY;
    double y = sy;
    double z = sz * cosY - sx * sinY;
    sx = x;
    sy = y * cosX + z * sinX;
    sz = z * cosX - y * sinX;
    x = sx * cosZ - sy * sinZ;
    y = sy * cosZ - sx * sinZ;
    if (sz + 1 > EPS)
    {
        x = x * MKOEF / (sz + 1)  + WINWIDTHD2;
        y = y * MKOEF / (sz + 1) + WINHEIGHTD2;
    }
    else
    {
        x = x * MKOEF * 1000 + WINWIDTHD2;
        y = y * MKOEF * 1000 + WINHEIGHTD2;
    }
    return Point3D(x , y , sz);
}
```

Вывод из технической части

В данном разделе были рассмотрены основные сведения о модулях программного обеспечения. Приводятся листинги реализаций алгоритмов.

4 | Исследовательская часть

В данном разделе будет проведены исследования программы, и проведен анализ полученных данных. Все эксперименты проводились на машине со следующими характеристиками:

- операционная система: Windows 10 64-разрядная операционная система;
- процессор: Intel(R) Core(TM) i3-7020U CPU @ 2.30GHz;
- оперативная память: 8,00 ГБ.

4.1 Первое исследование

Проведём исследование: как количество частичек огня в системе повлияет на производительность программы. Для этого для количества частиц огня от 0 до 2000 с шагом 100 замерим время одного рабочего цикла программы. Для повышения точности, измерения будут проводится несколько раз. В качестве результата возьмём среднее арифметическое всех замеров. Также, для наглядной оценки по данным эксперимента составим график зависимости время одного рабочего цикла от количества частиц огня.

4.2 Результаты первого исследования

Результаты замеров времени внесены в таблицу 4.1 и визуализированы на графике, рисунок 4.1.

Таблица 4.1 – Результаты первого эксперимента

Количество частиц	Время мс.
0	100
100	101
200	107
300	115
400	120
500	124
600	130
700	135
800	150
900	156
1000	160
1100	166
1200	173
1300	180
1400	185
1500	194
1600	202
1700	211
1800	217
1900	221
2000	230

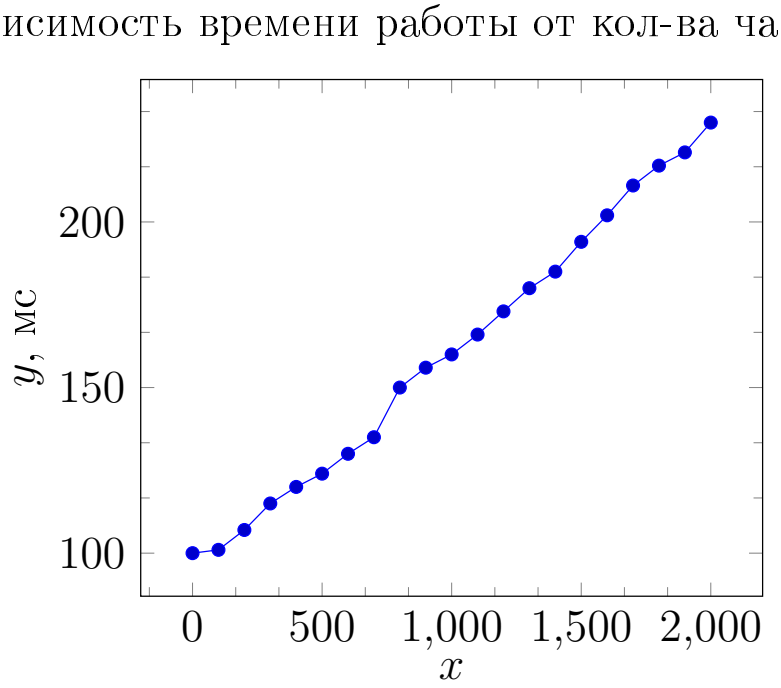

Рисунок 4.1 – График зависимости времени работы одного прохода цикла от количества частиц огня

4.3 Второе исследование

Проведём исследование: влияет ли тип модели топлива на скорость распространения огня. Для этого замерим время, прошедшее от задания пользователем первой точки до появления на модели 100 точек генерации огня. Для наибольшей точности проведём замеры по 3 раза.

4.4 Результаты второго исследования

Результаты замеров времени внесены в таблицу ??.

Таблица 4.2 – Результаты второго эксперимента

Тип материала	1 Замер мс.	2 Замер мс.	3 Замер мс.
Дерево	61290	67126	59149
Каменный уголь	34630	36089	39670
Древесный уголь	26062	26574	28940
Торф	52933	53155	54902

Вывод из исследовательской части

По результатам проведённых исследований можно сделать следующие выводы:

- зависимость времени одного цикла алгоритма от количества частиц огня линейная;
- тип модели влияет на скорость распространения пламени.

Заключение

В результате выполнения курсовой работы были выполнены все поставленные задачи. Было разработано и написано программное обеспечение. Были проведены исследования зависимости времени отрисовки от количества частиц огня и проверена зависимость скорости распространения огня от типа материала. По результатам исследований были сделаны выводы, что зависимость времени одного цикла алгоритма от количества частиц огня линейная и тип модели влияет на скорость распространения пламени.

Литература

[1] D. F. Rogers. Procedural Elements for Computer Graphics. 2nd ed., 1998 – p.457-517

[2] Никулин, Е.А. Компьютерная графика. Модели и алгоритмы: Учебное пособие / Е.А. Никулин. - СПб.: Лань, 2018. - 708 с.

[3] Шикин Е.В., Боресков А.В. Компьютерная графика. Полигональные модели. /М.: Диалог-МИФИ, 2000.

[4] Иванов, А.Е. Механика. Молекулярная физика и термодинамика: Учебник / А.Е. Иванов, С.А. Иванов. - М.: КноРус, 2016. - 320 с.

[5] Дегтярев, В.М. Компьютерная геометрия и графика: Учебник / В.М. Дегтярев. - М.: Академия, 2012. - 320 с.

[6] Документация по языку C++ [Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/ru-ru/cpp/cpp/?view=msvc-160https (дата обращения 21.10.13)