*Государственное образовательное учреждение высшего профессионального образования*

|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | ***«Московский государственный технический университет  имени Н.Э. Баумана»***  ***(МГТУ им. Н.Э. Баумана)*** |

ФАКУЛЬТЕТ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КАФЕДРА \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**РАСЧЁТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовой работе на тему:**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(Подпись, дата) (И.О.Фамилия)

Руководитель курсовой работы \_\_\_\_\_\_\_\_ **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(Подпись, дата) (И.О.Фамилия)

Москва, 20\_\_\_

СОДЕРЖАНИЕ.

Введение3

1. Аналитический раздел5
   1. Анализ предметной области5
   2. Классификация существующих методов5
   3. Обзор и анализ существующих программных систем, и обоснование необходимости разработки7
   4. Выбор, обоснование метода моделирования и алгоритма7
   5. Применяемые инструменты8
   6. Анализ компонент системы. Выделение абстракций9
2. Конструкторский раздел11
   1. Основы метода моделирования11
   2. Разработка алгоритма12
   3. Используемые типы и структуры данных13
   4. Диаграмма классов15
3. Технический раздел20
   1. Реализация алгоритма анимации. Огонь и дым20
   2. Реализация трехмерной сцены22
   3. Реализация статического окружения24
   4. Реализация интерфейса пользователя (GUI)24
4. Экспериментально-исследовательский раздел26
   1. Пользовательский интерфейс26
   2. Производительность ядра.28
   3. Дальнейшее развитие29

Заключение30

Список литературы31

ВВЕДЕНИЕ.

Краткая постановка задачи сформулирована в техническом задании проекта. Цель введения — раскрыть и дополнить техническое задание (ТЗ), а также выделить абстракции, понятия и сокращения, которые будут действительны на протяжении всего дальнейшего изложения.

Приведу формулировку технического задания:

«Разработать программу моделирования и анимации открытого огня. Программа должна реалистично визуализировать процесс горения на трехмерной сцене, при наличии статического окружения. Необходимо также реализовать анимацию дыма, получаемого в процессе горения и дать возможность пользователю настраивать параметры анимации».

Техническое задание описывает требования к программному продукту. Можно выделить шесть компонентов, из которых этот продукт должен состоять:

1) Трехмерная сцена.

2) Несложное статическое окружение.

3) Получение реалистичного огня (пламя, всполохи, спектры цветов …).

4) Рисование дыма, получаемого при горении.

5) Выделение ключевых параметров анимации.

6) Пользовательский интерфейс.

Связи между компонентами неочевидны и будут рассматриваться при построении архитектуры, но уже сейчас может быть выделена главная задача проекта — рендеринг реалистичной анимации. Все остальные задачи являются второстепенными и будут реализованы как расширения главной задачи, реализацию которой в программном коде, в дальнейшем будем называть ядром.

Можно видеть, что система разбита на два уровня: уровень ядра и уровень подзадач, в свою очередь разбитый на подзадачи. Большой плюс подобного разбиения в том, что его можно легко перенести на построение архитектуры программного решения. В таком случае, ядро и выделенные подзадачи становятся доменами.

Замечу, что здесь представлена вся рассматриваемая система, хоть и в самом абстрактном её понимании. Никаких других задач или уровней в дальнейшем вводиться не будет. То есть, вся последующая работа проводится вглубь обозначенных пунктов.

Расчетно-пояснительная записка поделена на разделы, каждый из которых описывает определенные аспекты проведенной работы:

- Анализ предметной области, выделение базовых программных сущностей, выбор инструментов разработки — находятся в аналитическом разделе.

- Математическая модель ядра описана и проанализирована в конструкторском разделе.

- Программная реализация составляет часть конструкторского раздела и полностью весь технический раздел.

- Тестирование и исследование программы вошли в экспериментальный раздел.

Терминология, принятая в данном документе не расходится с общепринятой в области компьютерной графики. Однако необходимо привести разъяснения по поводу слов «рендеринг» и «визуализация». В дальнейшем они будут употребляться как синонимы (второе чаще), в значении: процесс получения изображения по модели с помощью компьютерной программы.

2. АНАЛИТИЧЕСКИЙ РАЗДЕЛ.

**2.1. Анализ предметной области.**

В реальной жизни задача графической визуализации огня встречается достаточно часто. Однако проводить для этого реальные эксперименты дорого и опасно. Чтобы записать на видео красивый взрыв, необходимо выбрать удаленное от людей (а значит и от инфраструктуры) место, заготовить горючее и средства обеспечения безопасности, получить разрешение от властей. В итоге, полученные кадры будут эксклюзивными, ведь, чтобы переснять их в других ракурсах, потребуется повторение всего эксперимента. Тут и приходят на помощь программы-симуляторы.

Заказчиком технологии выступает в основном индустрия развлечений, которой огонь нужен в компьютерных играх и в кино, причем эти отрасли накладывают различные ограничения. Компьютерные игры требуют симуляции в реальном времени и с вычислительными ресурсами персонального компьютера. В то же время для киноиндустрии наиболее важна реалистичность. Ради получения реалистичных симуляций могут задействоваться большие вычислительные мощности и допустимо обрабатывать каждый кадр по несколько минут.

Министерство чрезвычайных ситуаций также использует программные симуляции природных явлений. Например, для таких задач как системы предсказания лесных пожаров и компьютерные системы обучения сотрудников.

В данной работе, решено сконцентрироваться на достижении довольно жестких требований, предъявляемых к компьютерным играм. Благодаря этому целевая аудитория созданного программного продукта будет наибольшей.

**2.2. Классификация существующих методов.**

Для рисования огня в компьютерной графике было изобретено достаточно много методов. Американский инженер H.R. Baum составил их классификацию в своей работе «Computer fire simulation»:

1) Particle-system based.

Огонь представляется в виде большого количества частиц, которые движутся по случайным траекториям, управляемые стохастическими процессами. Каждая частица имеет свой единственный цвет и некоторую степень прозрачности. При большом количестве частиц, метод сильно нагружает компьютер, при этом, не давая ни внешней, ни поведенческой реалистичности. Преимущество данного метода – простота реализации.

2) Noise-synthesized.

Здесь огонь получается путем наложения друг на друга нескольких сеток шумов. Выглядит хорошо, однако трудно воспроизвести реалистичное поведение.

3) Geometry-skeleton.

Используется лучевой каркас, который затем покрывается некоторой поверхностью. Данный метод подкупает своей простотой реализации и простотой расчетов. Других преимуществ не имеет.

4) Data-driven.

Допустим, имеется возможность наблюдать реальный огонь в реальном мире и фиксировать его с помощью фото или видеосъемки. Собранный материал можно использовать для построения компьютерной модели. Метод нуждается в большом хранилище данных и заведомо реалистичен. Однако чтобы добавить новое поведение огня, нужно снова проводить реальные эксперименты и увеличивать объем базы данных.

5) Physically-based.

К последнему классу относятся наиболее трудные для моделирования и программирования методы. Они же дают самый качественный результат, то есть самое реалистичное изображение. Поэтому, исследование «физически-обоснованных» анимаций представляет огромный интерес для всех, кто нуждается в искусственной симуляции огня на персональном компьютере. В данном направлении работают такие исследователи как Jos Stam, Ronald Fedkiw, Duc Quang Nguyen.

Недостатками метода являются сложная программная реализация, возможная расходимость применяемого метода расчета состояний модели и долгий расчет этой самой модели.

**2.3. Обзор и анализ существующих программных систем, и обоснование необходимости разработки.**

Для рендеринга огня могут быть использованы пакеты редактирования компьютерной графики: AutodeskMaya или Blender3D. Качество получаемых результатов для обоих приблизительно одинаково.

Blender3D – свободный пакет для создания трехмерной компьютерной графики, включающий в себя средства моделирования, анимации, рендеринга, постобработки видео, а также создания интерактивных игр. Имеет репутацию программы сложной для изучения.

AutodeskMaya – открытый для сторонних разработчиков редактор трехмерной графики. В настоящее время стал стандартом 3D графики в кино и телевидении.

Оба программных пакета включают в себя невероятно много функциональных возможностей, требуют знания специальных языков для настройки и работы (python для Blender, MEL основанный на Tcl, для Maya). Соответственно они требуют от пользователя долгого обучения, из-за чего использовать их могут лишь профессионалы либо специализированные студии. В то же время на рынке нет качественного инструмента, который могли бы использовать небольшие команды, разрабатывающие компьютерные игры или симуляторы.

Созданный программный продукт призван заполнить собой данную нишу.

**2.4. Выбор, обоснование метода моделирования и алгоритма.**

Для реализации ядра системы, автором был выбран Physically-based метод. Причиной выбора послужило то, что, хотя это и не самый простой метод, но он дает самый реалистичный и качественный результат. Также данный метод позволяет решать и некоторые дополнительные задачи, к примеру: распространение огня, тление и затухание. Это значит, что в будущем разработанная система может быть расширена до полноценного Фреймворка.

Существует занимательный факт, который несколько упрощает реализацию ядра. Дело в том, что движение жидкостей, газов и плазмы в поле скоростей можно описать одной и той же системой векторных уравнений. Система называется уравнениями Навье-Стокса, в честь французского физика Навье и британского математика Стокса. Конечно, так можно делать только тогда, когда не требуется физическая достоверность моделирования. Действительно, проектируемая модель используется для рендеринга, а не для проведения замеров физических характеристик и такой уровень точности просто не нужен.

Огонь, в первом приближении, является плазмой, а дым — газом. Значит оба явления можно описать с помощью одной и той же математической модели. Разница будет лишь в коэффициентах расчета, таких как вязкость или диффузия, цветовых спектрах и некоторых других. Замечу, что при реализации анимации огня другими методами, дым пришлось бы моделировать «с нуля».

Для реализации ядра в этой работе, был взят за основу программный движок, описанный JosStam в его статье Real-time fluid dynamics for games.

**2.5. Применяемые инструменты.**

Программа написана на языке С++ стандарта ISO/IEC 14882:2011. Это самая современная версия С++, которая на данный момент поддерживается компиляторами gcc-4.8 и выше. В программе использовались такие расширения языка как: синтаксис для лямбда-выражений, библиотека <function> с поддержкой функционального программирования и ключевое слово auto для выведения типов.

Для реализации оконного интерфейса и взаимодействия с графикой использовался программный инструментарий Qt. Среди причин выбора именно этой библиотеки можно назвать кроссплатформенность (программа собиралась и тестировалась на операционных системах Ubuntu Linux и Windows XP), богатый выбор легко расширяемых типов (в программе используются контейнерные классы QVector, QList, QMap, классы для создания графического интерфейса пользователя QMainWindow, QPainter и другие), а также механизм сигналов-слотов.

**2.6. Анализ компонент системы. Выделение абстракций.**

Представленный во введении набор компонент описывает главные задачи программного решения. Он был выделен из технического задания и компоненты не учитывали зависимости между задачами и метод моделирования. Поэтому они не могут быть напрямую использованы для построения ОО архитектуры.

В ходе объектно-ориентированного анализа некоторые из первоначальных компонент были объединены, какие-то остались без изменений, а третьи были разбиты на несколько компонент. В результате получился следующий список.

1. Пользовательский интерфейс – является прослойкой между приложением и пользователем. Он состоит из формы и элементов управления. Данный компонент работает с интерфейсом трехмерной сцены.

2. Трехмерная сцена – это компонент верхнего уровня, который формирует пространство объектов и содержит в себе такие абстракции как камера и экранная плоскость. Команды от пользовательского интерфейса перехватываются сценой и обрабатываются в ней.

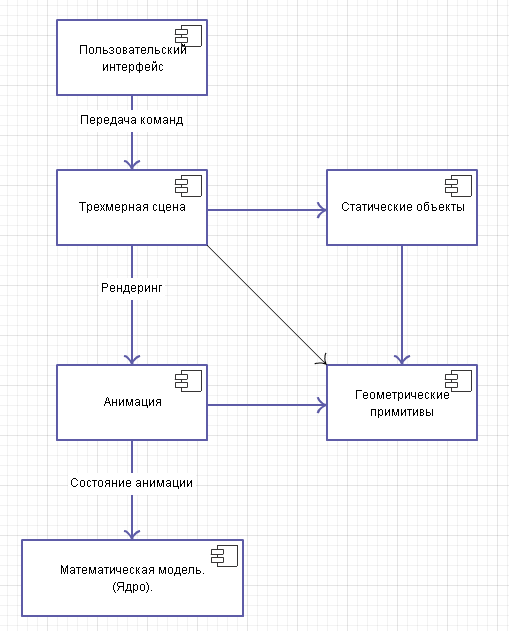
3. Статические объекты – находятся на сцене в заранее определенной локации. Не должны изменяться или анимироваться.

4. Анимация – компонент, который задает механику рендеринга огня и дыма. Именно здесь решается задача реалистичности изображения, определяются спектры цветов. Помимо этого, отдельный модуль данного компонента управляет настройками анимации: позволяет изменять их, сохранять в файлы и загружать. Настройки служат для изменения поведения анимации.

5. Ядро – реализация математической модели. Поверх ядра построена анимация.

6. Геометрические примитивы – компонент, представляющий собой низкий уровень реализации трехмерной схемы. Содержит в себе классы: точка, вектор, плоскость, полигон и операции с ними.

Зависимости компонент приведены на следующей диаграмме. Здесь стрелка обозначает использование того компонента, к которому она направлена тем, из которого она исходит.



Рисунок

Замечу, что перечисленные компоненты содержат в себе все задачи, выделенные при разборе технического задания. Также здесь заданы зависимости компонент друг от друга. Все вместе они представляют собой архитектуру программного решения.

3. КОНСТРУКТОРСКИЙ РАЗДЕЛ.

**3.1. Основы метода моделирования.**

Уравнения Навье-Стокса – система дифференциальных уравнений в частных производных для абстрактных векторных полей любой размерности, описывающая движение вязкой ньютоновской жидкости. Уравнения Навье-Стокса являются одними из важнейших в гидродинамике и применяются в математическом моделировании многих природных явлений и технических задач.

Пусть в момент времени t1 имеется n-мерная сетка, в каждой точке которой определены скорость U и плотность вещества R. Уравнения Навье-Стокса учитывают параметры среды и позволяют определить значения величин U, R для момента t2 = t1 + Δt.

Здесь и далее, полем называется векторное поле, на котором применимы операторы векторной алгебры (градиент, дивергенция, лапласиан) и определены параметры среды. Сетка – это математическая абстракция, описывающая поле с некоторой точностью. Первое понятие применяется при моделировании, а второе при реализации модели.

Скорость перемещения вещества в огне и дыме намного меньше, чем скорость звука. Следовательно, можно допустить, что скорость звука в рассматриваемых средах бесконечна и любые возмущения передаются немедленно. Данное условие позволяет использовать модель несжимаемой жидкости, что сильно упростит систему уравнений. Ведь если жидкость несжимаемая, то плотность сохраняется при изменении давления и само давление можно не принимать в расчет.

В результате, система уравнений модели выглядит следующим образом.

(1)

Это нелинейная система векторных уравнений в частных производных. Верхнее описывает поведение поля скоростей и называется уравнением движения. Оно в точности отражает эволюцию поля скоростей в течение времени. Нижнее уравнение описывает движение вещества в поле скоростей. Система справедлива как для двумерного, так и для трехмерного случая (меняется только количество компонент векторов).

Само по себе поле скоростей не имеет визуального смысла, пока нет объектов, на которые оно воздействует. Движение этих объектов рассчитывается путем перевода окружающих их скоростей в силы. Простые объекты, вроде частиц, обычно просто двигаются вдоль скоростей, но в случае сплошных сред моделировать каждую частицу слишком ресурсозатратно. Поэтому частицы вещества заменим плотностью: непрерывной функцией, которая для каждой точки пространства определяет количество вещества в ней.

Рассмотрим сначала второе уравнение. Оно утверждает, что изменение скалярного поля плотностей во времени происходит из-за трех причин, соответствующих трем слагаемым в правой части. Первая — это так называемая адвекция (advection), в сущности, перемещение вещества под действием поля скоростей. Вторая — диффузия вещества, где каппа — коэффициент диффузии. И, наконец, третья причина — поле источников плотности, обозначаемое S.

Верхнее уравнение очень похоже на нижнее. И действительно, три слагаемых в правой части имеют ту же самую смысловую нагрузку: адвекция, диффузия (которая в смысле скоростей является вязкостью), где ню — коэффициент, и источники поля (применительно для скоростей это поле сторонних сил; благодаря этому полю можно, например, моделировать ветер, сдувающий языки пламени).

К сожалению, в настоящее время не существует общего аналитического решения уравнений Навье-Стокса. Существование и гладкость решений этих уравнений – одна из семи математических задач тысячелетия, сформулированных в 2000 году Математическим институтом Клэя. По состоянию на 2014 год подтвержденные решения найдены лишь в некоторых частных случаях, обусловленных простой геометрией. В остальных случаях применяются численные методы.

**3.2. Разработка алгоритма.**

Математические уравнения и абстракции очень полезны при проектировании модели. Однако при реализации, необходимо перейти к конечным представлениям полей. Общепринятым подходом является использование конечных сеток, которые состоят из одинаковых ячеек. Внутри каждой ячейки характеристики поля являются постоянными величинами. Такое представление позволяет применить численные методы для решения уравнений Навье-Стокса, а также производить рендеринг на растровых дисплеях.

Замечу, что благодаря введению сеток мы получаем первый инструмент для оценки реалистичности получаемого изображения: величина равная отношению количества пикселей к количеству ячеек. Подробнее об этом будет сказано в экспериментальном разделе.

Для построения модели введем сетки размера N + 2 на N + 2 для полей скорости и плотности вещества. Помимо указанных, нам потребуются сетки для полей S (источники вещества) и f (сторонние силы) тех же размеров. Значения последних двух сеток не зависят от состояния модели, и не будут изменяться при решении системы (1). Каждая сетка имеет слой, толщиной в 1 ячейку, для задания граничных условий.

Ядро программного решения, по сути, является реализацией математической модели. Оно призвано решить систему Навье-Стокса для заданных сеток и коэффициентов диффузии и вязкости. За один запуск, алгоритм будет обходить сначала сетку плотностей и вычислять новые значения для неё при текущем фиксированном значении скоростей, а затем обходить сетку скоростей, чтобы учесть вязкость вещества и сторонние силы.

**3.3. Используемые типы и структуры данных.**

В программе используются три категории типов данных. Информация о наиболее важных представителях каждой категории сведена в таблицы.

Таблица

|  |  |  |  |
| --- | --- | --- | --- |
| **Тип данных** | **Назначение** | **Представление** | **Использующие классы** |
| FVal | Значение поля. | long | NS\_Grid, NS\_Solver. |
| Field | Сетка соотв. полю. | FVal \*\* | NS\_Grid, NS\_Solver. |
| Factor | Коэффициент при расчете сеток | FVal | NS\_Solver |

Первая категория – это примитивные типы языка С++, которые нужны для хранения и передачи простых, неструктурированных данных, а также в функциях и методах как локальные переменные.

Вторая категория – это типы, предоставляемые библиотекой Qt. В коде программы их легко можно узнать по префиксу Q в названии. В основном это контейнеры и типы для работы с графикой.

Таблица

|  |  |  |
| --- | --- | --- |
| **Тип данных** | **Назначение** | **Использующие классы** |
| QString, QVector | Контейнеры | UserInterface, Scene, Obstacle, Polygon3D |
| QPainter | Контекст рисования | Scene, Drawable, Fluid, Obstacle |
| QPoint | Точка на экране | Fluid, Point3D, Plane3D |
| QColor | Цвет RGBA | Fluid, FireGrid, SmokeGrid |
| QFile | Файловые запись и чтение | Adjustable |

Третья категория – это типы, созданные специально для данного программного решения. Среди них типы, представляющие собой геометрические примитивы, функциональные типы, для передачи специализированных анонимных функций и некоторые другие.

Таблица

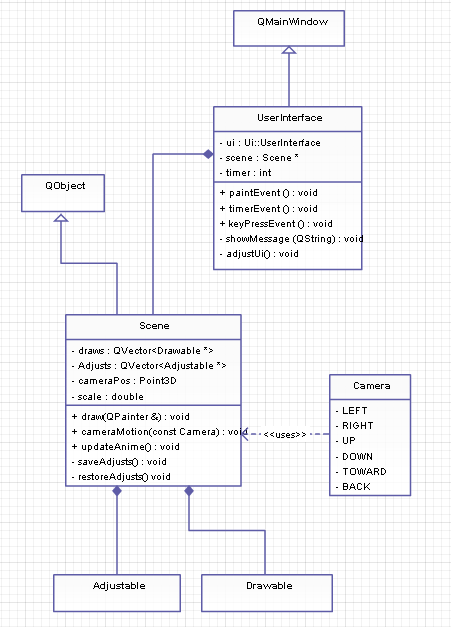
|  |  |  |  |
| --- | --- | --- | --- |
| **Тип данных** | **Назначение** | **Представление** | **Использующие классы** |
| Projector | Проецирующая функция | function <QPoint (Point3D &)> | Drawable, Fluid, Obstacle |
| Point3D,  Vector3D,  Polygon3D,  Plane3D | Геометрические примитивы для трехмерного пространства объектов | double X, Y, Z | Scene, Projector |
| QVector<Point3D> |
| double A, B, C, D |

**3.4. Диаграмма классов.**

Компонент системы может быть реализован одним из трех способов. Одним классом (Пользовательский интерфейс и сцена), несколькими классами (анимация, статические объекты и геометрические примитивы) или набором классов с выделенным пространством имен. Последний способ называется пакетным и используется ядром. Выбор того или иного представления зависит от сложности задач, которые решает компонент.

Особенность пакетного способа в том, что при импорте можно будет выбрать нужные сущности директивой using. Модули ядра содержат в себе много подробностей реализации, которые не нужны другим компонентам системы, и поэтому для них целесообразно ввести собственное пространство имен.

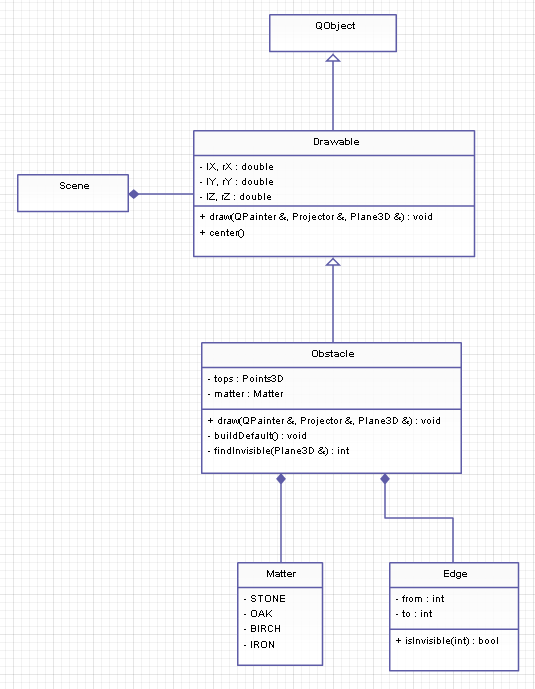
Диаграммы классов показывают внутреннее устройство компонент. Для классов указываются только самые важные атрибуты и методы. Здесь и далее используется нотация UML.



Рисунок

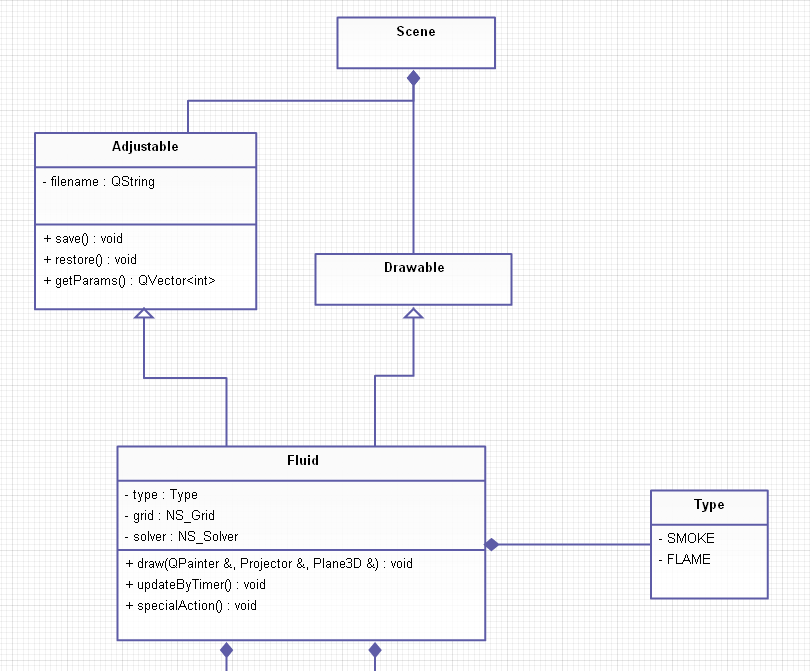
Конечно, привести полную диаграмму не получится, так как в ней слишком много классов. Поэтому диаграммы нарезаны кусками, в соответствии с компонентами программы. Первая часть соответствует устройству компонент «сцена» и «пользовательский интерфейс».

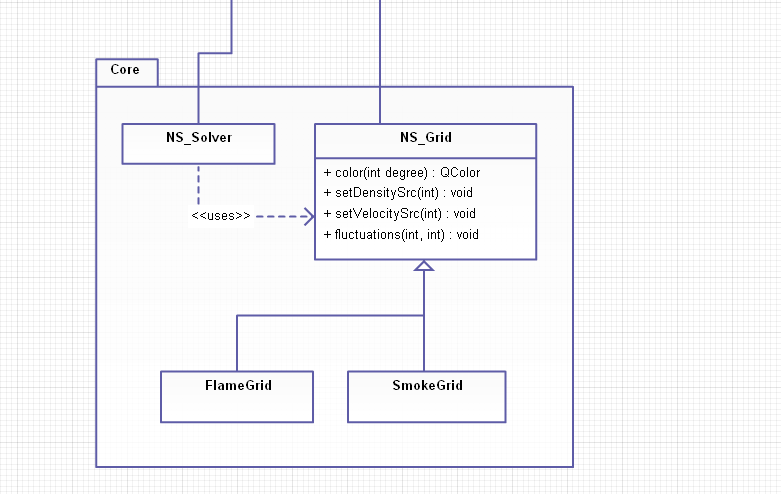
Для упрощения разработки архитектуры были применены паттерны проектирования. На первой диаграмме, представителем является паттерн «Фасад»: сцена предоставляет интерфейс для работы с приложением.



Рисунок

Вторая часть показывает часть устройства сцены (абстрактный класс Drawable) и компоненту статических объектов. Третья часть – компоненту анимации и ядро.





Рисунок

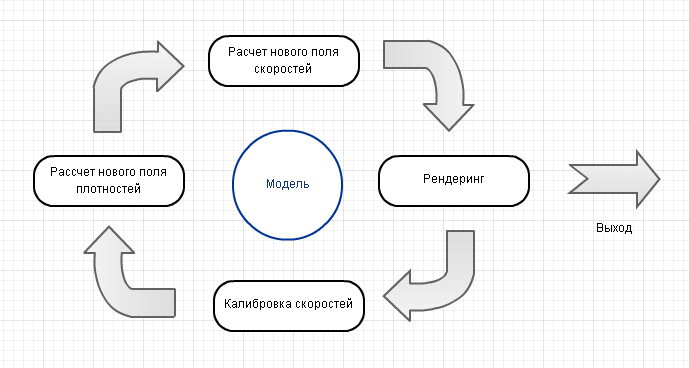
На третьей части диаграммы классов применяется такая техника как подмешивание функционала (mixin). Класс Fluid реализует интерфейс абстрактного класса Drawable и наследует класс Adjustable, для того чтобы получить возможность работы с сохранением и загрузкой настроек в файл.

Также примером подмешивания служит наследование QObject. Он позволяет задействовать механизм сигналов-слотов.

4. ТЕХНИЧЕСКИЙ РАЗДЕЛ.

**4.1. Реализация алгоритма анимации. Огонь и дым.**

Алгоритм анимации представляет собой безусловный цикл, каждая итерация которого соответствует смене кадра. Внутри цикла выполняется перерасчет полей плотности и скорости, а также визуализация огня и дыма на сцене. Помимо этого, производится калибровка поля скоростей, для того чтобы смоделировать действие температуры. Это необходимо делать, потому что температура не входит явно в систему уравнений (1). Цикл реализован в виде таймера Qt с задержкой в 41 миллисекунду, что дает частоту анимации равную 24 кадрам в секунду.



Рисунок

Задачей расчета новой сетки плотностей занимается класс NS\_Solver. В этом классе собраны методы, которые призваны смоделировать уравнения Навье-Стокса и решить их. Сами сетки и параметры вещества находятся в другом классе – NS\_Grid, который объявляет NS\_Solver как дружественный, чтобы предоставить ему возможность напрямую работать с сетками. Помимо этого NS\_Grid декларирует интерфейс для калибровки сеток и предоставляет методы для работы с плотностями для рендеринга. К примеру, чисто виртуальный метод

color(int degree) : QColor,

задает отображение плотности, соотнесенной к минимальному значениюв сетке, на цвет.

Класс NS\_Grid наследуется FlameGrid и SmokeGrid. Каждый из дочерних классов реализует интерфейс калибровки специфичный для своего вещества и метод color, задающий цветовой спектр анимации. Затем класс Fluid, конфигурируется нужным типом сети (см. диаграмму классов).

В интерфейс калибровки входят три метода.

* set\_dencity\_src(int)

Метод принимает на вход мощность источников вещества. При переопределении здесь задается конфигурация источников, специфичная для конкретной анимации.

* set\_velocity\_src(int)  
  То же для скоростей.
* fluctuations(int, int)  
  Метод вносит элемент случайности в поведении анимации. Первый параметр задает компоненту X случайных изменений, а второй – компоненту Y.

Реализация математической модели сокрыта в реализации класса NS\_Solver. Методы этого класса моделируют три процесса (и для скорости и для плотности), соответствующие трем слагаемым в правой части системы (1). Это диффузия, адвекция и учет поля источников. Последнее реализуется наиболее просто. Мы просто суммируем сетку поля с сеткой источника.

field[I][J] += dt \* source[I][J];

Здесь впервые появляется параметр dt. Это один из самых важных параметров, так как от него зависит, будет ли анимация огня выглядеть как огонь или как набор вспыхивающих и носящихся точек. По сути, этот параметр задает скорость распространения вещества.

Для учета диффузии NS\_Solver в каждой точки сети рассчитывает, сколько вещества ушло из ячейки и сколько вещества пришло от четырех соседей. Расчет учитывает коэффициент диффузии.

float a = dt \* diff \* N \* N;

x[i,j] = (x0[i,j] + a\*(x[i-1,j] + x[i+1,j] + x[i,j-1] + x[i,j+1])) / (1 + 4\*a);

Реализация адвекции основана на разложении Ходжа (Hodge decomposition) и построена в соответствии со способом, изложенным в статье Jos Stam.

**4.2. Реализация трехмерной сцены.**

Трехмерная сцена является абстракцией предметной области и на уровне кода представлена классом Scene. Внутри этого класса задаются камера и объекты сцены. Ради упрощения системы для камеры не было предусмотрено собственного объекта. Внутри сцены, камера реализована в виде точки cameraPos (x, y, z), которая соответствует центру экранной плоскости.

Для задания экранной плоскости, помимо позиции камеры, используются ещё две точки: cameraX и cameraY. Дополнительные точки указывают направление осей координат экранной плоскости в трехмерном пространстве объектов. При старте программы экранная плоскость инициализируется как

int userZ = 1000;

cameraPos = Point3D(0, 0, userZ);

cameraX = Point3D(1, 0, userZ);

cameraY = Point3D(0, 1, userZ);

scale = 1;

Для того чтобы поменять ракурс необходимо применить одно из доступных преобразований ко всем трем точкам. В программе предусмотрено два преобразования: вращение относительно оси Oy на фиксированный угол и вращение относительно оси Ox. Преобразование выполняется относительно центра сцены (0, 0, 0).

Для того чтобы смоделировать масштабирование, программа использует коэффициент масштабирования scale. Преобразование из пространства объектов в экранную плоскость учитывает scale чтобы растягивать или сжимать изображение..

scale \*= zoom;

Vector3D singleX = (cameraX - cameraPos) \* scale;

Vector3D singleY = (cameraY - cameraPos) \* scale;

Все объекты сцены являются наследниками абстрактного класса Drawable, который задает интерфейс рисования, положение и габариты объекта. Внутри Drawable используется прямоугольная оболочка. Такое решение позволяет упростить способ рендеринга, ведь для прямоугольных оболочек достаточно использовать алгоритм художника.

for (Drawable \* draw : draws) {

dist = plane.dist(daw->center());

dists.append(dist, draw);

}

qSort(dists.begin(), dists.end(),

[] (A, B) { return A.dist > B.dist; }

);

Перед визуализацией, все объекты сортируются в соответствии с величиной расстояния до экранной плоскости. Чем меньше расстояние, тем позже объект должен быть отрисован. Затем происходит визуализация объектов.

Для того чтобы перенести реализацию рендеринга из сцены в сами классы, наследующие Drawable используется проецирующая функция.

typedef function<QPoint (const Point3D &)> Projector;

Это функция специального вида, которая передаётся как параметр методу рендеринга конкретного класса. Она получает на вход точку (x, y, z) из пространства объектов и преобразует её в точку на экранной плоскости, учитывая при этом положение камеры и масштабирование.

**4.3. Реализация статического окружения.**

В программе реализован всего один вид статического окружения. Это класс Obstacle («препятствие»), представляющий собой твердотельный неанимируемый параллелепипед. Его положение и материал задаются сценой при инициализации.

При создании объекта «препятствия» можно задать его материал. В зависимости от материала будет выбрана текстура для закрашивания объекта при визуализации. Все файлы текстур находятся в специальном хранилище – ресурс Qt. Его преимущество перед складированием текстур в явном виде в том, что при сборке проекта Qt запакует файлы ресурсов в исполняемый файл. Таким образом, ресурсы не могут быть потеряны либо забыты.

Объект «препятствия» способен самостоятельно определить, какие ребра и грани являются невидимыми и не станет их визуализировать.

QPoints Ps = projector(tops);

int invisible = findInvisible(plane);

for (Edge & edge : edges)

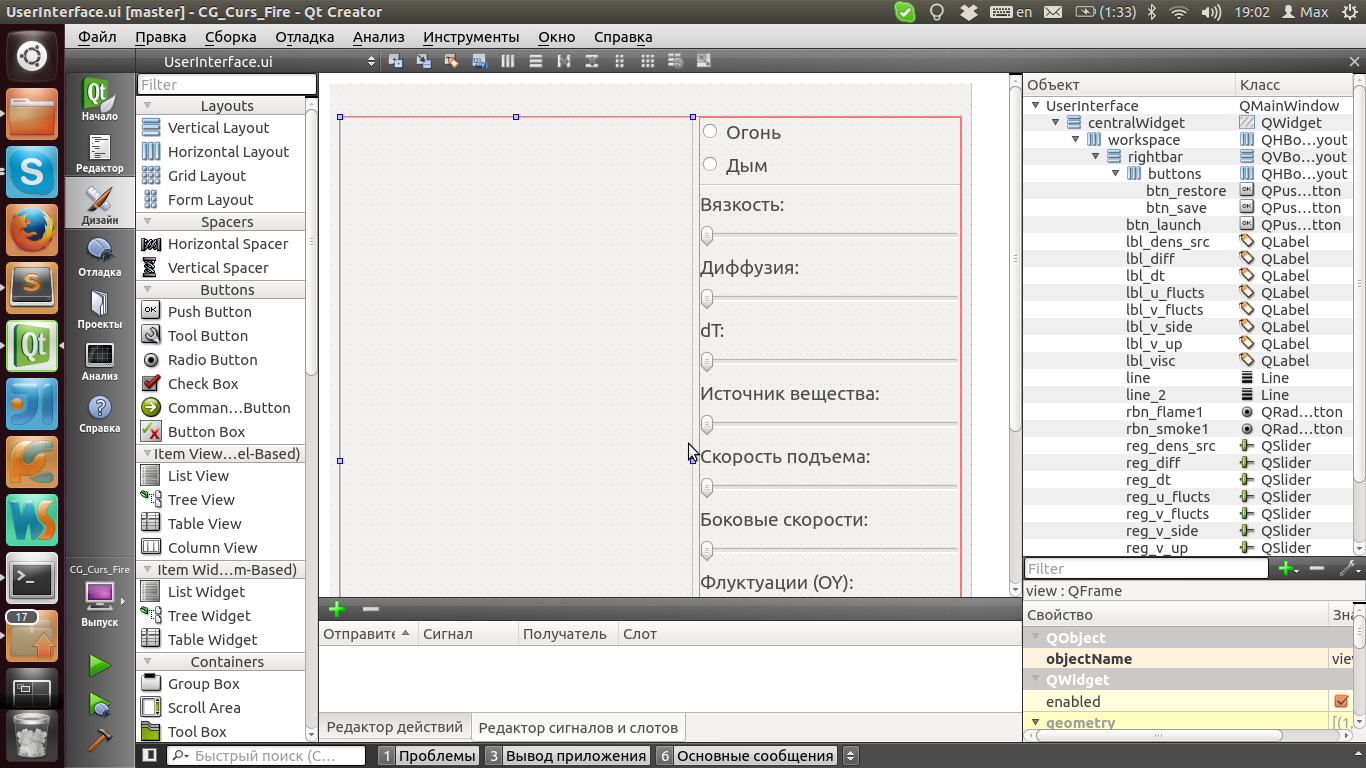
if (! edge.isInvisible(invisible))

painter.drawLine(Ps[edge.from], Ps[edge.to]);

Здесь метод findInvisible анализирует, какая вершина параллелепипеда самая далекая от экранной плоскости и возвращает номер этой вершины в массиве Ps.

**4.4. Реализация интерфейса пользователя (GUI).**

Для создания пользовательского интерфейса использовался инструмент быстрого прототипирования Qt Designer. Инструмент позволяет строить диалоговые окна из виджетов, компоновать и настраивать их.



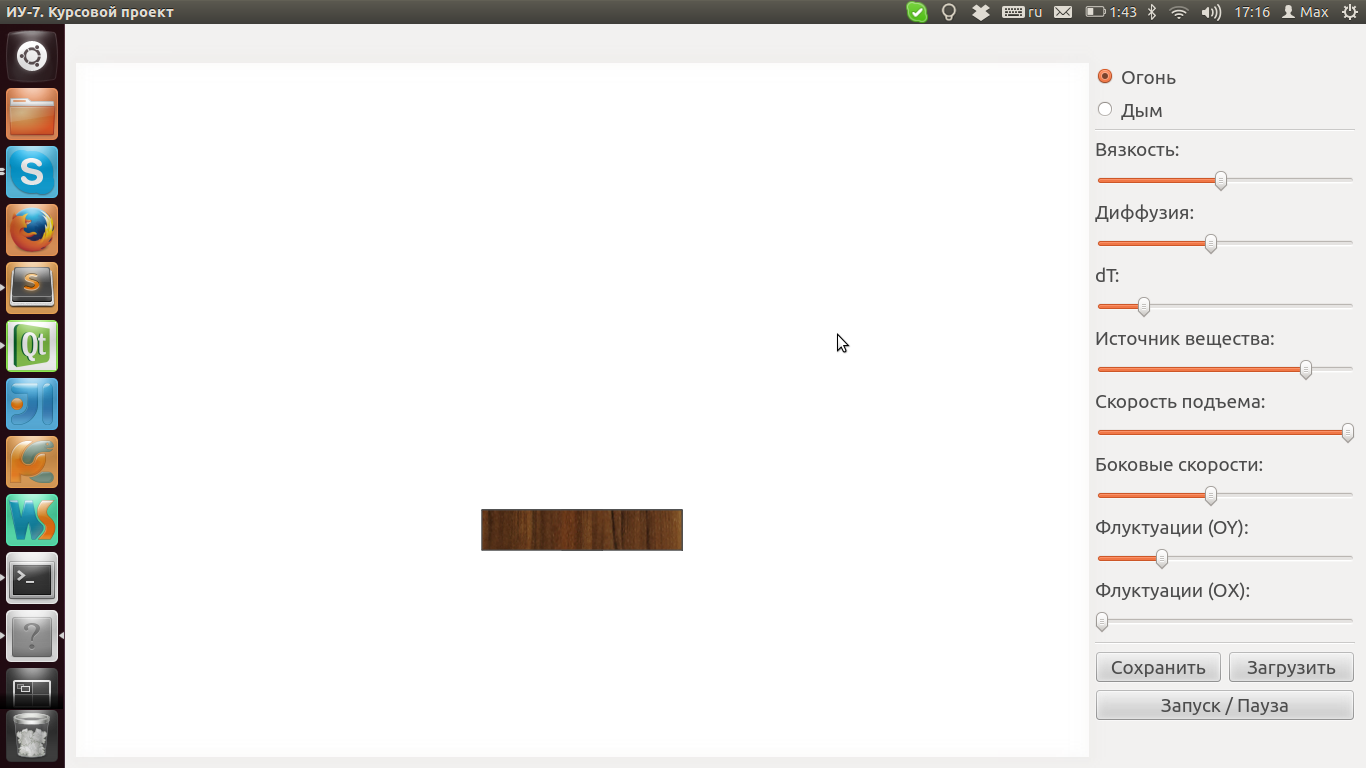
Рисунок

В самой программе доступ к сверстанной форме осуществляется через специальный объект Ui. Этот объект управляется из класса UserInterface.

5. ЭКСПЕРИМЕНТАЛЬНО-ИССЛЕДОВАТЕЛЬСКИЙ РАЗДЕЛ.

**5.1. Пользовательский интерфейс.**

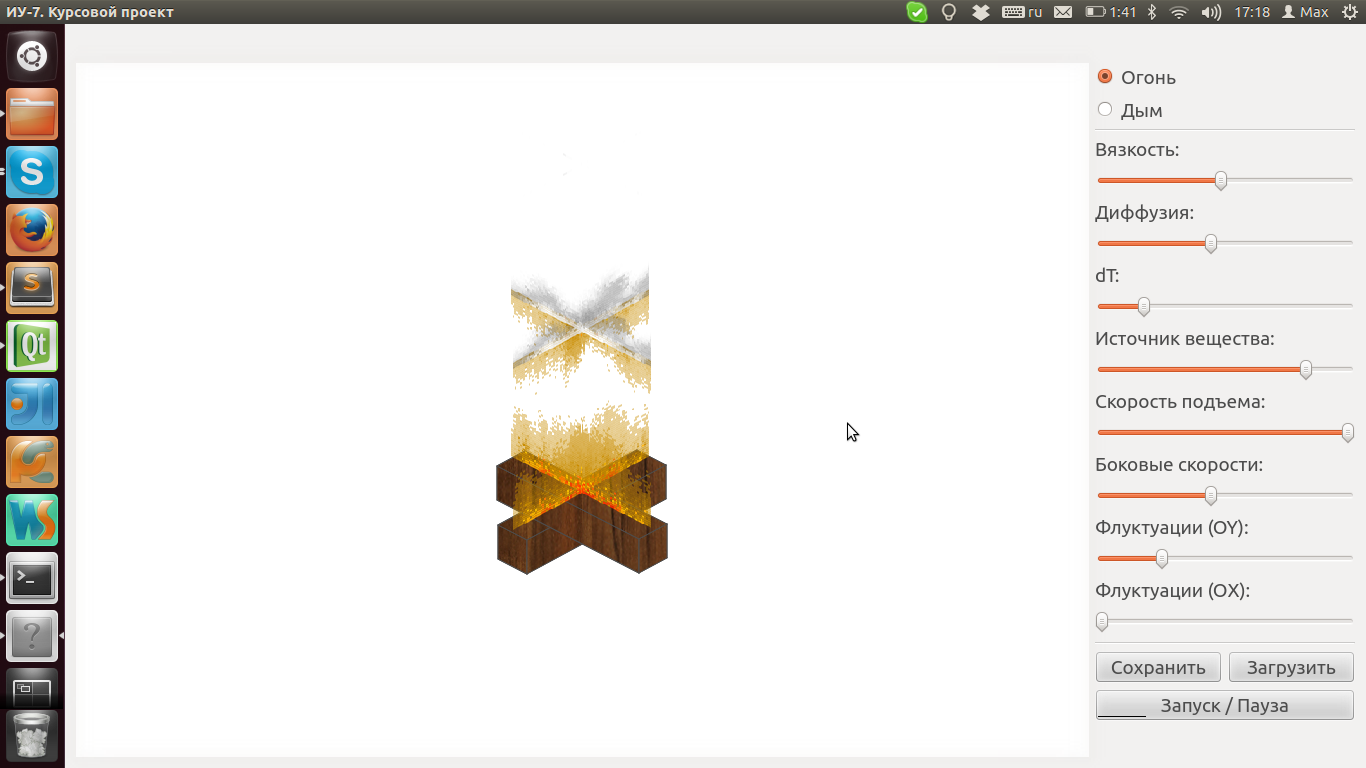
На следующем рисунке представлен внешний вид интерфейса программы при запуске. Анимация не начинается до тех пор, пока пользователь не подаст явной команды, нажав на кнопку запуска.



Рисунок

В правой части интерфейса находится управляющая панель. На неё вынесено управление параметрами анимации. Параметры можно изменять, используя ползунки, сохранять в файл и загружать из файла. Для последних двух действий предназначены специальные кнопки внизу. Перед изменением какого-либо параметра выберете, для огня или для дыма изменение будет применено.

Анимацию также можно ставить на паузу.

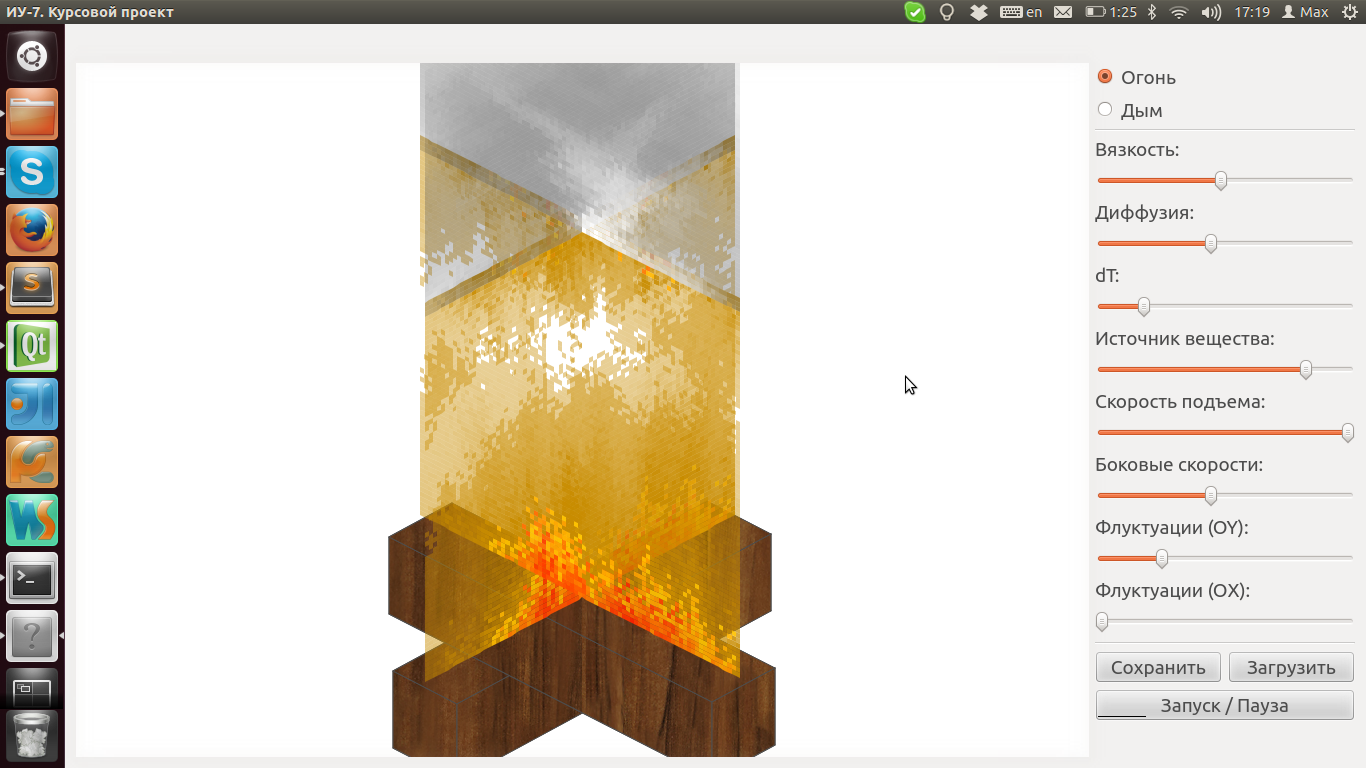


Рисунок

Для управления камерой используйте кнопки на клавиатуре.

* W – вращает камеру oтн. оси Ox вверх.
* S – вращает камеру oтн. оси Ox вниз.
* A – вращает камеру oтн. оси Oy влево.
* D – вращает камеру oтн. оси Oy вправо.
* Z – приближает камеру.
* X – отдаляет камеру.

Управление камерой не вынесено на управляющую панель, чтобы не перегружать интерфейс. После старта анимации можно выбрать нужный ракурс.



Рисунок

**5.2. Производительность ядра.**

Реалистичность анимации очень сильно зависит от скорости работы ядра и визуализации. Программа должна перерисовывать экран 24 кадра в секунду, а значит необходимо оптимизировать самые затратные операции. В данной работе такой операцией является перерасчет модели.

Хорошо известно, что целочисленные математические операции выполняются в компьютере гораздо быстрее, чем операции над числами с плавающей точкой. Поэтому, чтобы ускорить систему, ядро было переписано на целочисленную арифметику. Вместо типа float используется тип long (4 байта = 4 байта). Операции умножения и деления новых чисел заменяются макросами:

typedef long FVal;

#define FPP 9

#define X1\_0 (1<<FPP)

// преобразования для FVal

#define I2X(x) ((FVal)((x)<<FPP))

#define F2X(x) ((FVal)((x)\*X1\_0))

#define X2I(x) ((int)((x)>>FPP))

#define X2F(x) ((float)(x)/X1\_0)

// замены для операций умножения и деления

#define XM(x, y) ((FVal)(((x)\*(y))>>FPP))

#define XD(x, y) ((FVal)(((x)<<FPP)/(y)))

Теперь все сетки содержат значения типа long, а не float. Точность представления новых чисел фиксирована и равна 9 знакам после запятой.

Ускорение работы действительно было достигнуто. Для сеток размером N = 40 ячеек один кадр обрабатывался не 48 мс, а 30 мс.

**5.3. Дальнейшее развитие.**

На данный момент программа имеет один большой недостаток. Расчет и визуализация анимации производятся центральным процессором в однопоточном режиме. Такое решение работает очень медленно и из-за этого вряд ли будет в неизменном виде применяться на практике.

Обычно графические приложения создают для работы напрямую с видеокартой. Для таких целей существуют специальные пакеты программ, например OpenCL и Cuda. Их использование могло бы повысить скорость работы программного решения на порядки. Однако реализация становится сложной и выходит за рамки данной курсовой работы.

В качестве дальнейшего развития проекта можно назвать замену алгоритма художника на алгоритм Z-буфера. Это повысит качество рендеринга и позволит отказаться от ограничений прямоугольной оболочки.

Нереализованной осталась ещё одна хорошая идея. Для повышения реалистичности анимации можно было бы задать связь между горючим материалом, огнем и дымом. К примеру, рассчитывать поле источников дыма в соответствии с полем плотностей огня.

ЗАКЛЮЧЕНИЕ

Реализованный программный продукт позволяет получать реалистичные анимации огня и дыма. При реализации использовались алгоритмы машинной графики, такие как алгоритм художника.

Программа построена так, что, изменяя параметры, можно получать различное поведение и вид анимации.

Программный продукт может быть расширен и преобразован в полноценный Фреймворк для моделирования различных природных явлений, например, облаков, осадков, огня, тумана, дыма, течения жидкостей.

Однако, несмотря на все преимущества, все-таки остаются и недостатки. Используется не самый лучший метод удаления невидимых поверхностей. Производительность анимации может быть улучшена при реализации на видеокарте.

В заключение хотелось бы заметить, что в результате курсовой работы было произведено большое количество исследовательской работы, как в области машинной графики, так и в области математического моделирования и программирования в целом. Благодаря данной работе, автор смог приобрести большое количество новых знаний и навыков, что, несомненно, пригодится ему в будущем.

**Список литературы.**

1. Роджерс Д . Алгоритмические основы машинной графики : Пер . с агл . – М .: Мир , 1989. – 512 с.
2. Jos Stam, "Real-Time Fluid Dynamics for Games". Proceedings of the Game Developer Conference, March 2003
3. Гради Буч и др. Объектно-ориентированный анализ и проектирование с примерами приложений (UML 2). Третье издание = Object-Oriented Analysis and Design with Applications (3rd Edition). — М.: «Вильямс», 2010. — 720 с. — ISBN 978-5-8459-1401-9.
4. Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес Приемы объектно-ориентированного проектирования. Паттерны проектирования = Design Patterns: Elements of Reusable Object-Oriented Software. — СПб: «Питер», 2007. — С. 366. — ISBN 978-5-469-01136-1. (также ISBN 5-272-00355-1)
5. Макс Шлее Qt 4.8 Профессиональное программирование на C++. — СПб.: «БХВ-Петербург», 2012. — С. 912. — ISBN 978-5-9775-0736-3.
6. Стивен Прата - Язык программирования C++. Лекции и упражнения, 6-е изд. : Пер. с англ. М. : ООО «И.Д. Вильямс» 2013 г. ISBN: 978-5-8459-1778-2