МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ   
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования

УРАЛЬСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

имени первого Президента России Б. Н. Ельцина

ИНСТИТУТ ЕСТЕСТВЕННЫХ НАУК И МАТЕМАТИКИ

Кафедра высокопроизводительных компьютерных технологий

**ВИЗУАЛИЗАЦИЯ СЕТОЧНЫХ ДАННЫХ С ПРИМЕНЕНИЕМ РЕДАКТОРА УЗЛОВ**

Направление подготовки 09.04.03 «Прикладная информатика»

|  |  |
| --- | --- |
| Зав. кафедрой:  д. ф.-м. н., с.н.с. М.Ю. Филимонов | Магистерская диссертация  **Вяткина**  **Дениса Евгеньевича**  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Нормоконтролер:  к. ф.-м. н. В.С. Зверев  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Научный руководитель:  д. ф.-м. н., проф.  О. В. Ушакова  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Екатеринбург

2021

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

УРАЛЬСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

имени первого Президента России Б.Н. Ельцина

ИНСТИТУТ ЕСТЕСТВЕННЫХ НАУК И МАТЕМАТИКИ

УТВЕРЖДАЮ

Руководитель образовательной программы

( \_)

«1» декабря 2020 г.

Код, наименование направления: 09.04.03 Прикладная информатика

Наименование программы: Прикладная информатика в аналитической экономике

Группа: МЕНМ-290901

**ЗАДАНИЕ**

на выполнение выпускной квалификационной работы

студента Вяткина Дениса Евгеньевича

(Фамилия, имя, отчество)

Квалификация: магистр

(бакалавр, специалист, магистр)

Провести научное исследование по теме: Визуализация сеточных данных с применением редактора узлов

Срок представления работы научному руководителю: «9» июня 2021 г.

Научный руководитель (Ушакова О.В.)

(Подпись) (И.О. Фамилия)

Задание принял к исполнению

(Подпись)

# РЕФЕРАТ

Вяткин Д.Е., ВИЗУАЛИЗАЦИЯ СЕТОЧНЫХ ДАННЫХ С ПРИМЕНЕНИЕМ РЕДАКТОРА УЗЛОВ, квалификационная работа на степень магистра: 49 с., 20 рис., 21 источник, 2 приложения.

Ключевые слова: научная визуализация, компьютерная графика, расчётные сетки, материал, редактор узлов.

Предмет исследования — трёхмерные структурированные расчётные сетки, используемые для численного решения дифференциальных уравнений методом конечных разностей.

Цель работы — проектирование и реализация графического пакета для визуализации расчётных сеток, с возможностью расширения пользователями, без необходимости дополнительного программирования.

Результат работы — графический пакет и технические средства для визуализации расчётных сеток с применением визуального редактора узлов, позволяющего создавать новые визуальные представления без дополнительного программирования.

Vyatkin D., VISUALIZATION OF NUMERICAL GRID DATA USING NODE EDITOR, master’s degree thesis: 49 pages, 20 pictures, 21 references, 2 appendices.

Key words: scientific visualization, computer graphics, numerical grids, material, node editor.

Subject of research — three-dimensional structured numerical grids used in numerical methods for solving differential equations with finite difference method.

Goal of the work — design and implement graphics software for numerical grid visualization, expendable by user without additional programming of the system.

Result of the work — graphics software and technologies for numerical grid visualization using node editor that lets users create new visual configurations without additional programming.

# МЕСТО ВЫПОЛНЕНИЯ РАБОТЫ

Место выполнения работы Уральский Федеральный Университет имени первого Президента России Б.Н. Ельцина, институт естественных наук и математики, кафедра высокопроизводительных компьютерных технологий.

**СОДЕРЖАНИЕ**

[РЕФЕРАТ 2](#_Toc74087816)

[МЕСТО ВЫПОЛНЕНИЯ РАБОТЫ 3](#_Toc74087817)

[ВВЕДЕНИЕ 6](#_Toc74087818)

[ОСНОВНАЯ ЧАСТЬ 8](#_Toc74087819)

[1 Обзор литературы 8](#_Toc74087820)

[1.1 Общие положения о расчетных сетках 8](#_Toc74087821)

[1.2 Предыдущие результаты в области визуализации сеток 9](#_Toc74087822)

[2 Постановка задачи работы 10](#_Toc74087823)

[2.1 Актуальность и необходимость визуализации расчётных сеток 10](#_Toc74087824)

[2.2 Анализ потребностей пользователей 11](#_Toc74087825)

[2.3 Цели и задачи работы 12](#_Toc74087826)

[3 Способы и методы решения задачи 13](#_Toc74087827)

[3.1 Организация рабочего процесса 13](#_Toc74087828)

[3.2 Обсуждение идей, предложенных на первой итерации 14](#_Toc74087829)

[3.3 Формат хранения данных 15](#_Toc74087830)

[3.4 Выбор технологий разработки 16](#_Toc74087831)

[3.5 Базовое отображение данных 19](#_Toc74087832)

[3.6 Управление камерой 22](#_Toc74087833)

[3.7 Автоматическая перезагрузка шейдеров 23](#_Toc74087834)

[3.8.1 Концепция материалов 24](#_Toc74087835)

[3.8.2 Организация параметров материала 25](#_Toc74087836)

[3.8.3 Настройка параметров материалов в пользовательском интерфейсе 26](#_Toc74087837)

[3.8.4 Пример материала 28](#_Toc74087838)

[3.9 Визуальное создание материалов 30](#_Toc74087839)

[3.9.1 Концепция редактора узлов 30](#_Toc74087840)

[3.9.2 Описание редактора узлов 32](#_Toc74087841)

[3.9.3 Генерация шейдеров 33](#_Toc74087842)

[3.9.4 Библиотека узлов-операций 34](#_Toc74087843)

[3.9.4 Сохранение и загрузка материалов 35](#_Toc74087844)

[3.9.5 Организация сцены 36](#_Toc74087845)

[3.10 Результаты и их обсуждение 37](#_Toc74087846)

[3.10.1 Примеры отображений 37](#_Toc74087847)

[3.10.3 Направления доработок и улучшений 43](#_Toc74087848)

[ЗАКЛЮЧЕНИЕ 44](#_Toc74087849)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ 45](#_Toc74087850)

[ПРИЛОЖЕНИЕ А 47](#_Toc74087851)

[ПРИЛОЖЕНИЕ Б 49](#_Toc74087852)

# ВВЕДЕНИЕ

Расчётные сетки применяются для аппроксимации различных объектов в рамках решения уравнений математической физики методом конечных разностей. Результаты находят активное применение в практических инженерных задачах.

Ставится вопрос визуализации и интерпретации получаемых данных. Качественное специализированное решение для визуализации сеточных данных необходимо для обеспечения наглядной и удобной работы исследователей — прикладных математиков.

Существует ряд программных пакетов, в какой-то степени решающих поставленный вопрос. Но они не удовлетворяют полностью особым требованиям учёных университета по различным причинам, которые в конце концов сводятся к невозможности гибко расширять возможности пакета под собственные нужды.

Цель данной работы — проектирование и реализация графического пакета для визуализации расчётных сеток с возможностью добавления новых визуальных представлений, необходимых пользователю, без дополнительного программирования. Цель достигается в рамках выполнения ряда формулируемых задач, реализуемых в виде экспериментов, направленных на добавление и изучение возможностей различных концепций и функций.

Обозначенный графический пакет является отдельным приложением для персональных компьютеров, реализованным на языке программирования C++ с применением библиотеки OpenGL для выведения графических данных на экран. Для дополнительной функциональности также использован ряд дополнительных библиотек, находящихся в открытом доступе.

Реализованные эксперименты основаны на возможности в реальном времени перезагружать шейдеры — программы, выполняемые на графических ускорителях. Изучаются разные подходы к определению новых шейдеров и взаимодействию с ними через пользовательский интерфейс.

В частности, на одном из этапов работы были адаптированы результаты, полученные при разработке многих популярных программ для 3D-моделирования и разработки компьютерных игр. Речь идет про концепцию материалов и дополняющую её концепцию редактора узлов, ставшую, в конце концов, основным результатом работы.

Определение новых визуальных отображений с помощью визуального редактора делает порог входа для новых пользователей достаточно низким, позволяет им сконцентрироваться на решении своих задач.

Рассмотрен поэтапный процесс приближения к результатам работы, а также их техническая реализация. Описана работа редактора узлов, приведены примеры визуальных отображений. В приложении приведена ссылка для получения исходного кода из открытого источника и требования для компиляции программы.

# ОСНОВНАЯ ЧАСТЬ

# 1 Обзор литературы

**1.1 Общие положения о расчетных сетках**

Расчетная сетка — совокупность узлов, заданных в области определения некоторой функции, аппроксимирующих область для применения численных методов. В общем виде построение сетки означает построение непрерывного отображения из простой вычислительной области на более сложную физическую. Примеры расчётных сеток можно увидеть на рисунках 1.1 и 1.2.

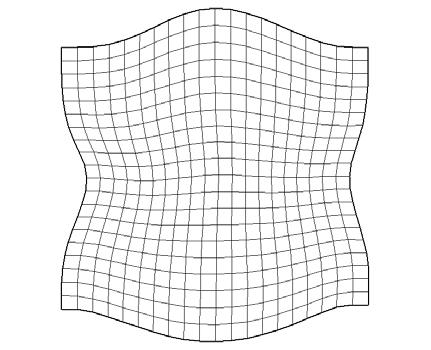


Рисунок 1.1 – Пример двумерной расчетной сетки

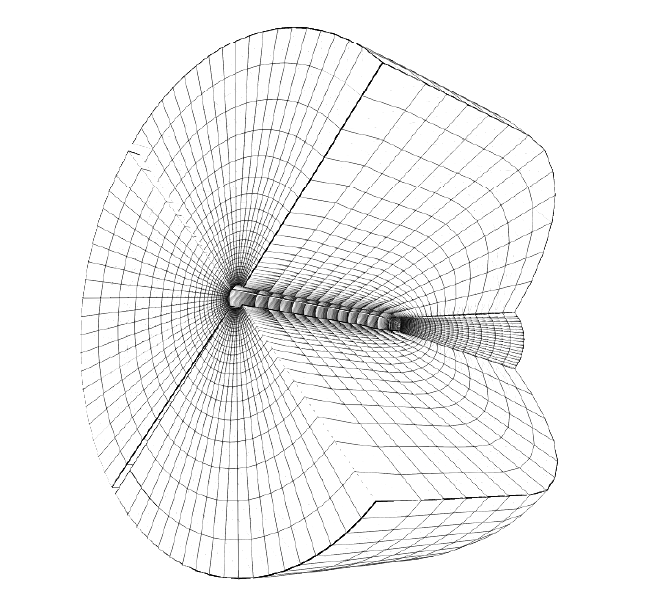


Рисунок 1.2 – Пример трехмерной расчетной сетки

Для решения дифференциальных и интегральных уравнений зачастую применяются численные методы, в том числе метод конечных разностей, который заключается в замене производных разностными схемами. Преимущество — высокая производительность, достигаемая за счет параллельных вычислений.

Среди всех сеток выделяют неструктурированные и структурированные. Условие структурированности сетки — существование отношения порядка на множестве узлов, которое задается кусочно-гладким отображением из параметрического пространства с целочисленными координатами узлов в физическое пространство.

Также расчетные сетки разделяют на вырожденные и невырожденные. Невырожденность сетки — важнейший критерий, который приходится учитывать в исследованиях. Сетка является невырожденной, если отображение вычислительной области в физическую — биекция. Это условие исключает наличие самопересечений, зазоров и выходов за пределы расчетной области в сетке в физической области.

Невырожденность сетки зависит от значений якобиана применяемого отображения в узлах сетки. Якобиан вычисляется как определитель матрицы частных производных отображения. Неположительные значения якобиана позволяют говорит о вырожденности сетки [19].

Основные теоретические сведения по расчётным сеткам были получены в рамках прохождения курса «Методы построения сеток», проведенного осенью 2019 года Ушаковой О.В.

**1.2 Предыдущие результаты в области визуализации сеток**

В рамках магистерской диссертации Анненковой О.Г. была разработана система визуализации сеточных данных, ориентированная на научных сотрудников [1].

В работе изучены перспективы использования веб-среды в качестве основы системы. Разработанная программа запускается в браузере и позволяет загружать данные с удаленного сервера. Графическая визуализация обеспечена при помощи библиотеки WebGL.

Программа является модулем для системы Viewlang, разработанной Васёвым П.А. Viewlang определил особенности организации сцены и реализации некоторых алгоритмов, применяемых в разработанной программе [2].

Также в рамках данной диссертации были изучены вопросы реализации поддержки виртуальной реальности и управления жестами, направленных на погружение пользователя в абстрактную среду. Упомянута важность обращения внимания на психофизиологический фактор виртуальной реальности, на обеспечение комфортного восприятия пользователем моделируемой среды [13].

Набор реализованных функций включает в себя разбор сеточных данных в предоставленном формате, фильтрацию, разреживание и выборку групп ячеек по индексам. Присутствует возможность экспорта иллюстраций.

Отличие данной работы от перечисленных результатов заключается в изучении способов обеспечения гибкости и расширяемости набора функций программы пользовательскими средствами.

# 2 Постановка задачи работы

**2.1 Актуальность и необходимость визуализации расчётных сеток**

Задачи, решаемые с применением метода конечных разностей и расчётных сеток, регулярно возникают в ходе проведения прикладных исследований в области математической физики. Расчётные сетки в данном случае представляют собой аппроксимацию некоторого реального физического объекта.

В ходе работы формируется набор данных, включающий в себя описание структуры сетки, расположение всех её узлов, а также опциональную дополнительную информацию о свойствах объекта. Возможность визуального отображения и интерпретации полученных данных крайне важна для исследователей. Визуализация должна позволять научным работникам понять, просмотреть и получить представление о данных.

Существует ряд коммерческих и некоммерческих графических пакетов, в функциональность которых включена возможность отображения сеточных данных. Среди них можно выделить системы Tecplot [18] и ParaView [14], каждая из которых обладает очень широким набором функций. Тем не менее, у обеих систем есть проблемы, не позволяющие получить необходимые математикам университета визуальные представления данных. Tecplot — коммерческий продукт, исходный код которого не распространяется в свободном доступе, поэтому пользователи ограничены реализованными функциями. ParaView — сложный и универсальный инструмент общего назначения, недостаточно удобный для работы с конкретными данными.

Из описанной выше ситуации можно сделать вывод, что визуализацию расчётных сеток нельзя считать решенной задачей и на рынке программного обеспечения есть пространство для новых продуктов, избегающих проблем упомянутых программных пакетов.

**2.2 Анализ потребностей пользователей**

Потребности пользователей не ограничиваются применением только уже существующих функций в доступных графических пакетах. В общих словах можно сформулировать следующую глобальную проблему: у пользователя нет возможности реализовать новое визуальное представление, либо реализация окажется слишком трудозатратной.

Реализации новых визуальных представлений могут помешать следующие проблемы:

1. исходный код программного продукта не находится в открытом доступе;
2. реализация новых функций требует знания системных языков программирования, программной инженерии и других специальных знаний;
3. работа с программным продуктом сложна и неудобна сама по себе, что делает его расширение нецелесообразным в долгосрочном плане;
4. разработка новых функций занимает много времени и отвлекает от решения непосредственных задач пользователя;
5. отсутствует возможность повторного использования однажды реализованной функции; нельзя комбинировать существующие функции.

Исходя из приведенных факторов можно сформулировать требования к новому графическому пакету, решающему указанную проблему. Вместо создания системы, обладающей определенным набором возможностей, следует разработать расширяемую средствами пользователя платформу, не требующую для этого дополнительного программирования самой системы.

**2.3 Цели и задачи работы**

Цель работы — проектирование и реализация графического пакета для визуализации расчетных сеток, с возможностью расширения пользователями, без необходимости дополнительного программирования системы.

Для достижения цели необходимо выполнение следующих задач:

1. изучить формат хранения сеточных данных, с которым предстоит работать;
2. выбрать набор технологий для разработки системы;
3. реализовать базовый функционал для отображения сеточных данных;
4. провести ряд экспериментов, связанных с созданием возможности расширять функционал системы без изменения исходного кода и повторной компиляции программы;
5. определить оптимальный вариант из испробованных в ходе экспериментов и использовать его в качестве итогового;
6. выложить исходный код в открытый доступ, описать процесс добавления новых функций;
7. определить и описать возможные направления дальнейшего развития разработанной системы.

# 3 Способы и методы решения задачи

**3.1 Организация рабочего процесса**

Процесс разработки программного продукта был построен на основе гибкой методологии разработки. Работа разбита на относительно короткие итерации, в начале каждой из которых формируются новые требования к программе, а к концу эти требования должны быть реализованы. В рамках каждой итерации корректируется курс развития проекта с учетом последних полученных результатов. Таким образом, сама программа и функциональные требования к ней могут поменяться радикальным образом после любой из итераций.

Среди достоинств такого подхода можно выделить возможность избегать ситуации, когда большое количество времени было потрачено на реализацию функций, которые в итоге не решают поставленные задачи. Возможность гибко менять направление развития проекта позволяет сохранять связь с реальными потребностями пользователей.

Главным недостатком подхода можно назвать пренебрежение долгосрочным планированием, которое может приводить к накопление технических недостатков, которые в будущем могут помешать внесению новых изменений.

В случае обсуждаемой в рамках этой работы программы данный подход — допустимый и полезный, так как одной из поставленных задач является проведение ряда экспериментов, которые должны помочь определить оптимальные способы достижения цели работы. Отсутствие четкого понимания того, какие функции необходимы программе, не позволяет разработать долгосрочный план развития, что оправдывает применение гибкой методологии разработки.

Важная часть работы — получение обратной связи на каждой итерации. В данной работе обратная связь была организована в рамках проведения научных семинаров на базе сектора визуализации ИММ УрО РАН, где автор работы показывал текущее состояние разрабатываемого программного обеспечения, а ряд присутствующих специалистов мог задать интересующие их вопросы и предложить новые идеи. Предложенные идеи стали основным источником прогресса, позволившим на каждой итерации приближаться к решению поставленных задач.

**3.2 Обсуждение идей, предложенных на первой итерации**

В начале работы были предложены следующие идеи для реализации в специализированном программном обеспечении, разрабатываемом в рамках работы над магистерской диссертацией:

1. визуализация общего представления о сетках на базе имеющихся данных;
2. поддержка сеток, содержащих в себе более миллиона ячеек. Должна сохраняться высокая производительность;
3. анализ особенностей, их выделение цветом или формой;
4. поддержка виртуальной реальности с применением специализированных гарнитур. Возможна реализация таких видов отображения, как общий облет и движение над сеткой по пространственным координатам, погружение в сетку с детализацией информации о ее внутренности и выделением отдельных сеток, навигация по сетке с помощью некоторых ориентиров, движение по сетке с изменением направления и скорости;
5. поддержка с помощью различных контроллеров и естественных жестовых интерфейсов.

Так как большая часть работы проводилась на личном оборудовании автора в рамках обязательного дистанционного обучения, не получилось организовать доступ к необходимому оборудованию для работы с виртуальной реальностью. В результате анализа потребностей пользователей было принято решение сфокусироваться на задачах, описанных в подразделе 3.3, так как обозначенные проблемы показались наиболее значимыми и фундаментальными для построения системы. Реализация отложенных задач по-прежнему возможна по мере необходимости и появления доступа к специализированному оборудованию.

**3.3 Формат хранения данных**

В качестве образцов сеточных данных был предоставлен набор файлов в текстовом формате Tecplot ASCII [18]. В данном наборе каждая сетка — результат преобразования трёхмерной прямоугольной сетки. Ячейки включают в себя по восемь узлов.

Каждый файл содержит описание набора блоков, на которые разбита расчетная сетка. Описание блока включает в себя:

1. строку с названием данного блока;
2. строку с перечислением используемых переменных. Она включает в себя трехмерные координаты узлов, а также любые другие значения, которые исследователь может приложить к данным;
3. строку с указанием размерностей блока;
4. набор строк, содержащих значения для каждой из используемых переменных. Количество равно произведению размерностей блока.

Узлу соответствует уникальный набор индексов. Узлы, у которых только один из индексов отличается на единицу, являются соседними и соединены дугами.

Пример данных:

TITLE = "str\_data"

VARIABLES = "X", "Y", "Z", "D", "P", "M", "S",

ZONE I= 26, J= 18, K= 9, F=POINT

-3.4740e-01 7.5072e-02 -6.8791e-02 5.1091e-01 4.1617e-01 2.4685e+00 1.0656e+00

-3.6450e-01 7.9131e-02 -7.2510e-02 6.0631e-01 5.2775e-01 2.3225e+00 1.0633e+00

-3.8108e-01 8.1687e-02 -7.4853e-02 1.0448e+00 1.1139e+00 1.8682e+00 1.0476e+00

-3.9749e-01 8.2730e-02 -7.5808e-02 1.0928e+00 1.1826e+00 1.8305e+00 1.0444e+00

...

Перед реализацией загрузки данных были рассмотрены существующие решения с открытым исходным кодом. В итоге было принято решение написать собственный простейший разбор данных. Так как формат данных достаточно простой, это не заняло много времени, при этом написанный код получился компактным и легко изменяемым.

В данный момент каждый раз при загрузке сетки производится полный разбор текстового файла. Для сеток относительно небольших размеров по 10-15 мегабайт время загрузки может составлять 4-5 секунд. На данном этапе эти значения можно считать приемлемыми. В будущем, если возникнет потребность в загрузке очень большого количества данных, можно реализовать сохранение загруженных в первый раз данных в бинарный формат, что позволит в следующие запуски сэкономить время на открытии сетки.

**3.4 Выбор технологий разработки**

Для отображения графических данных существуют специализированные аппаратные ускорители для отображения графики — GPU [16]. Из-за объема сеточных данных потребность в применении вычислений на GPU при разработке программного обеспечения очевидна. Работа с GPU подразумевает выбор некоторого графической библиотеки, через которую организуется общение между вычислительным процессором CPU и GPU по принципу клиент-сервер [10]. В качестве такой графической библиотеки была выбрана OpenGL. Среди её достоинств можно выделить поддержку большинства современных платформ и языков программирования и открытый исходный код.

Было принято решение о разработке приложения для персональных компьютеров. В качестве языка выбран системный язык C++ [17]. Данный язык — наиболее распространенный в сфере современной компьютерной графики, для него существует очень подробная документация и написано множество полезных вспомогательных библиотек. Также этот язык позволяет достичь максимальной производительности, так как отсутствуют промежуточные уровни выполнения кода, такие как сборщик мусора и интерпретатор.

Для работы с операционной системой выбрана популярная в игровой индустрии библиотека Simple DirectMedia Layer (SDL) [15]. В ее возможности входит поддержка различных платформ, создание окна операционной системы, создание контекста для работы с GPU с помощью различных графических библиотек, обработка ввода с мыши, клавиатуры и других контроллеров.

Для создания пользовательских интерфейсов выбрана библиотека Dear ImGUI, построенная на технологии Intermediate Mode Graphical User Interface [6]. Эта технология заключается в императивном подходе к объявлению структуры и функционала пользовательского интерфейса вместо более распространенного в данный момент декларативного подхода. Библиотека специализируется на применении в приложениях по визуализации данных. Принцип ее работы позволяет легко добавлять новые элементы во время разработки и динамически изменять состояние интерфейса во время работы программы. Пример кода, использующего библиотеку:

ImGui::Text("Hello, world %d", 123); // Форматированная надпись

if (ImGui::Button("Save")) // Кнопка сохранения

MySaveFunction(); // Функция сохранения

ImGui::InputText("string", buf, IM\_ARRAYSIZE(buf)); // Поле ввода

ImGui::SliderFloat("float", &f, 0.0f, 1.0f); // Слайдер

Результат выполнения кода можно увидеть на рисунке 3.1. Видно, что в результате каждого вызова процедуры из библиотеки Dear ImGUI отображается некоторый элемент интерфейса. Такой принцип работы стал важным фактором, позволившим провести предстоящий ряд экспериментов.

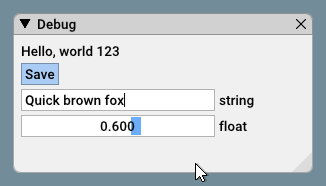


Рисунок 3.1 – Пример интерфейса, созданного с помощью библиотеки Dear ImGUI

Разработка велась в среде разработки Visual Studio 2019 [20]. Для организации контроля версий был создан репозиторий (хранилище исходного кода и других данных, связанных с проектом) с помощью системы Git [8]. О том, как получить исходный код и запустить его на личном компьютере, можно узнать в приложении Б.

Для некоторых второстепенных функций были использованы несколько дополнительных библиотек. Для редактора узлов, описанного в одном из следующих подразделов, применено расширение для библиотеки Dear ImGUI — imgui-node-editor [9]. Форматирование строк реализовано с помощью библиотеки fmt [7]. Для сохранения и загрузки структур данных использована библиотека ajson [3].

**3.5 Базовое отображение данных**

Перед тем, как приступить к реализации экспериментов, необходимо было реализовать базовую возможность отображать некоторую сетку, представленную в описанном ранее формате.

Когда сеточные данные загружены из файла, их необходимо представить в подходящем для OpenGL виде. Для это необходимо выполнить следующую последовательность действий:

1. создать Vertex Array Object (VAO) [21]. Это внутренняя структура данных OpenGL, которая хранит в себе связи с другими структурами, отвечающими за разные части процесса отображения данных. Таким образом, VAO помогает лучше организовать работу с данными в программе;
2. для каждого блока сетки создать Vertex Buffer Object (VBO) [21]. Эта структура хранит в себе непрерывный участок памяти с произвольными данными. В данном случае это координаты узлов, значения параметров, индексы. После создания VBO необходимо привязать к VAO.
3. для каждого VBO создать Index Buffer Object (IBO) [21]. Эта структура содержит список индексов элементов буфера из VBO. Она указывает, в каком порядке нужно отображать узлы. Это позволяет организовать отображение узлов, соединенных рёбрами, так как каждый узел может быть присоединен к нескольким дугам. Для каждой дуги достаточно записать в список IBO пару индексов узлов. IBO также необходимо привязать к VAO.
4. настроить атрибуты для VBO [21]. В данном случае атрибуты — части данных о сетке, среди них позиции узлов и параметры, добавленные автором сетки. Настройка включает в себя описание структуры области памяти, которую хранит VBO. Необходимо указать, где начинается и заканчивается каждый тип передаваемых данных [4].

После выполнения описанных действий, конфигурация данных для отображения завершена. В цикле приложение необходимо сделать активным VAO, созданный для загруженной сетки и вызвать специальную функцию для отображения графики.

Но сначала следует определить, каким образом отображаются предоставленные данные. Для этих целей используются шейдеры. Шейдер — программа, исполняемая на GPU [2]. Существует различные типы шейдеров, применяемые на разных этапах отображения графики. Каждый тип шейдера работает с определенным типом данных.

В разработанной программе используются два типа шейдеров: вершинный и пиксельный. Вершинный шейдер определяет положение каждого узла сетки. Пиксельный шейдер определяет цвет каждого пикселя экрана.

Для того, чтобы применить некоторый набор шейдеров, необходимо загрузить их тексты из файлов, скомпилировать средствами OpenGL и объединить в шейдерную программу. Тогда эту программу можно использовать для отображения данных.

Написание шейдеров происходит с применением специализированного языка программирования GLSL [4]. Пример вершинного шейдера:

layout(location = 0) in vec4 inPosition; // Координаты узла

layout(location = 1) in float inParam; // Произвольный параметр

out float param; // Передаем параметр в пиксельный шейдер

uniform mat4 view; // Матрица вида

uniform mat4 projection; // Матрица проекции

void main(void)

{

param = inParam;

gl\_PointSize = 2.0f; // Задаем размер точки

// Определяем позицию относительно камеры

gl\_Position = projection \* view \* inPosition;

}

Пример пиксельного шейдера:

in float param; // Переданный из вершинного шейдера параметр

out vec4 outColor; // Итоговый цвет пикселя

uniform vec3 defaultColor; // Настраиваемый цвет по умолчанию

void main(void)

{

outColor = vec4(defaultColor.r, defaultColor.g, defaultColor.b, 1.0) \* param; // Задаем цвет в зависимости от параметра

}

Визуальное представление расчётной сетки можно увидеть на рисунках 3.2 и 3.3. Была реализованы функции отображения отдельных узлов в виде точек и рёбер между узлами в виде линий.

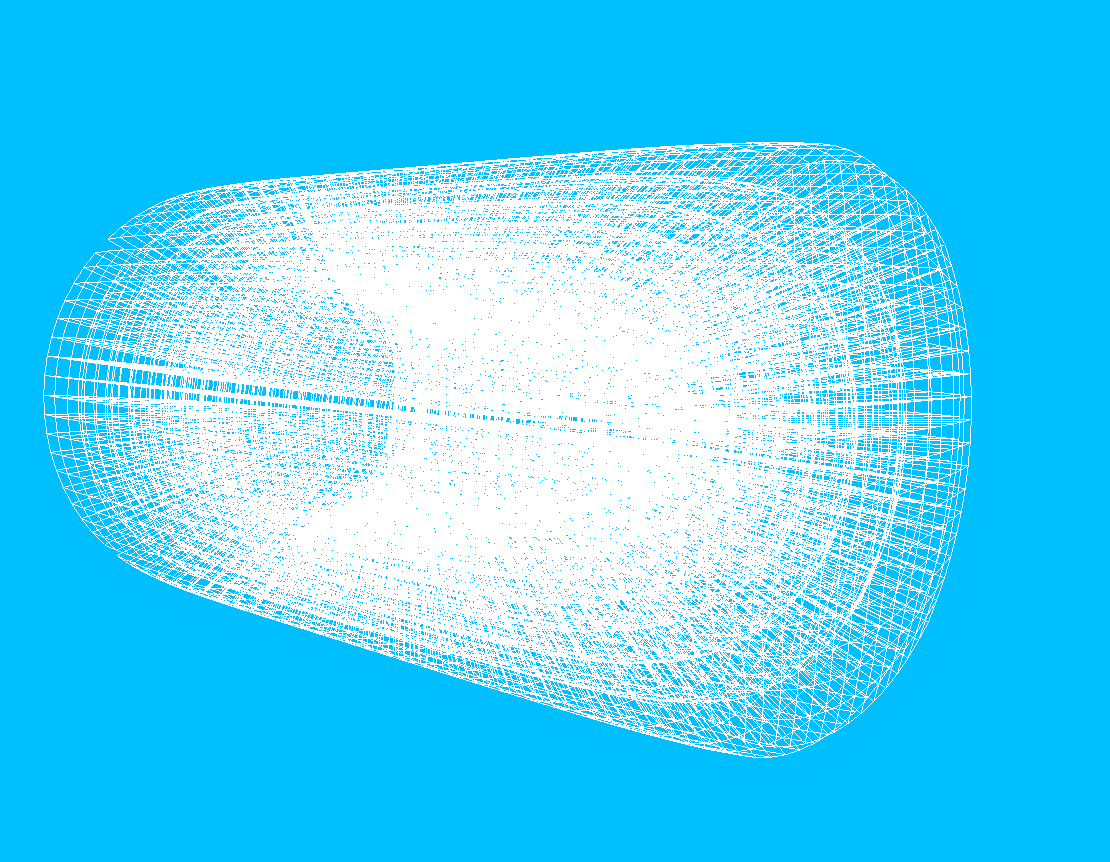


Рисунок 3.2 – Отображение сеточных данных в виде узлов, соединенных дугами

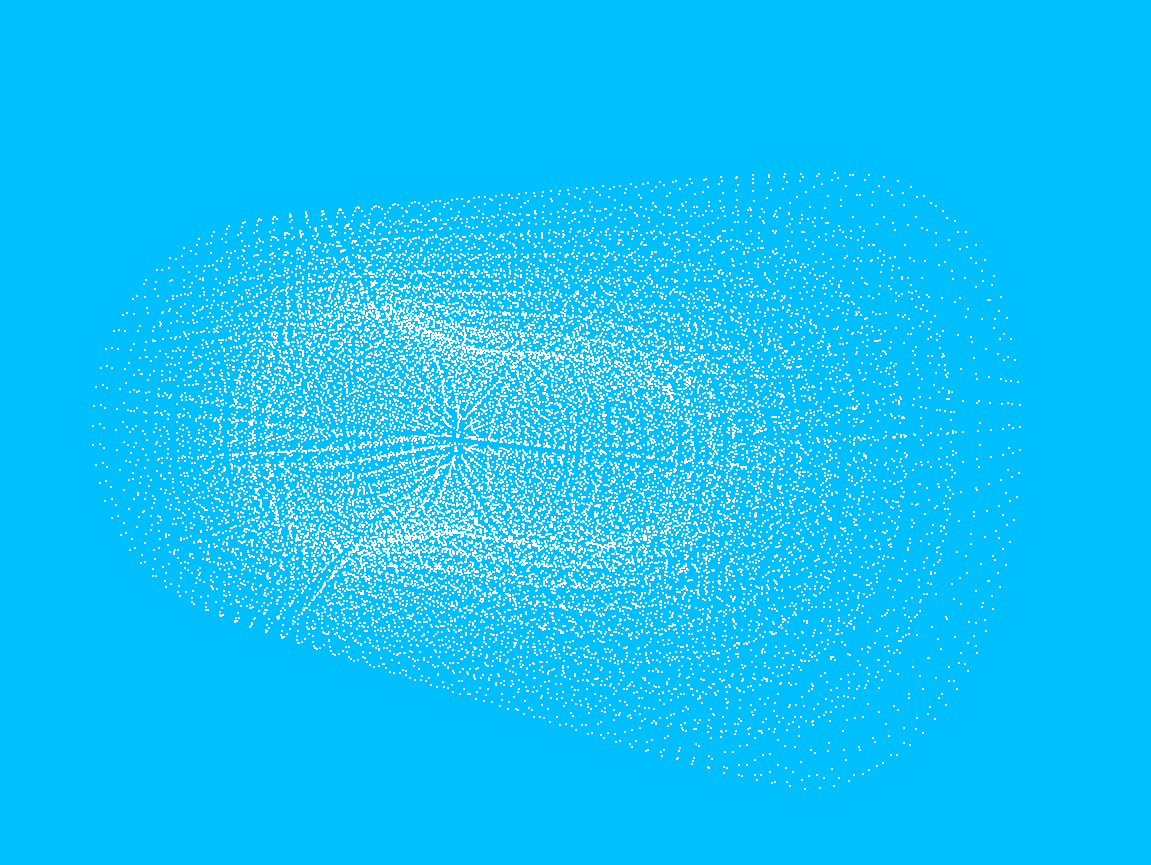


Рисунок 3.3 – Отображение сеточных данных в виде отдельных узлов

**3.6 Управление камерой**

Для изучения сеточных данных с различных ракурсов необходима возможность управлять положением камеры.

Были реализованы два типа управления:

1. свободная камера — с помощью мыши осуществляется поворот камеры вокруг собственного центра по двум осям. С помощью клавиатуры можно перемещать камеру в любом направлении. Этот тип управления подходит для перемещения внутри сетки [12];
2. орбитальная камера — все управление осуществляется с помощью мыши. Посредством движения мыши камера поворачивается вокруг заранее вычисленного центра сетки. Позиция камеры задается сферическими координатами, которые затем переводятся в декартовы [12].

Логичный путь развития системы управления просмотром сетки — подключение поддержки виртуальной реальности в комбинации со специальными контроллерами. Тем не менее, полезно сохранить базовый подход для пользователей без необходимого оборудования или имеющих проблемы с восприятием виртуальной реальности.

**3.7 Автоматическая перезагрузка шейдеров**

Возможность, реализованная в первом из проведенных экспериментов, изначально был направлена только на повышение удобства разработки и не предназначена для пользователей пакета. Но в итоге она стала ключевой для всех последующих экспериментов.

Эта возможность — автоматическая перезагрузка шейдеров во время работы программы.

Очень важным фактором для продуктивной работы с программой является скорость, с которой можно увидеть результат вносимых изменений. Каждое изменение алгоритма для визуального представления материала влечет за собой необходимость перекомпиляции шейдеров, применяемых в материале. Ручная компиляция не очень удобна и требует постоянного повторения однообразных действий.

Для решения этой проблемы была реализована динамическая подгрузка шейдеров. Она включает в себя следующие этапы:

1. раз в секунду производится проверка последнего времени изменения файлов, содержащих исходный код шейдеров. Если время изменения оказалось позже последней компиляции шейдера, необходимо полностью перекомпилировать шейдерную программу;

2. при необходимости перекомпиляции загружается исходный код шейдеров из соответствующих файлов, каждый шейдер компилируется по отдельности;

3. отдельные скомпилированные шейдеры связываются в одну шейдерную программу, которая загружается в память видеокарты;

4. заново формируется связь сеточных данных и параметров материала с шейдерной программой. С этого момента можно видеть визуальный результат внесенных изменений.

Описанная функция позволила быстро пробовать реализовывать разные визуальные представления данных, просто меняя исходный код шейдера и сохраняя файл. Возникло предположение, что подобный процесс также мог бы подойти и для конечных пользователей, так как изменение и добавление новых шейдеров не требует перекомпиляции самого графического пакета. Язык GLSL обладает гораздо более лаконичным и простым синтаксисом, чем C++, изучить его можно достаточно быстро и уже через пару часов написать простой шейдер для отображения сетки.

**3.8 Отображение расчётных сеток с помощью материалов**

**3.8.1 Концепция материалов**

Описанная выше возможность привела к появлению идеи о том, как можно её обобщить и сделать более удобной для пользователей. Речь идет о концепции материалов, которая встречается во многих программах для работы с 3D-графикой, не связанных с областью научной визуализации. Материалы активно используются в программных пакетах, применяемых в кинематографе, компьютерных играх, художественной компьютерной графике.

Материал представляет собой комбинацию из алгоритма, определяющего визуальное представление объекта, и структуры данных, хранящей параметры для указанного алгоритма. Алгоритм может быть задан с помощью классического языка программирования, шейдера или же с помощью визуального программирования. Значения параметров материала чаще всего задаются через пользовательский интерфейс [4].

Реализация концепции материалов включает в себя следующие подзадачи:

1. реализовать базовый синтаксический анализ шейдеров для определения набора параметров, используемых в шейдере;
2. реализовать динамическое отображение параметров шейдера в пользовательском интерфейсе;
3. реализовать передачу настраиваемых через интерфейс параметров в память графического ускорителя.

В полноценно реализованной системе материалов обязательно присутствует поддержка сохранения и загрузки полученных результатов в файлы. Это позволяет делиться своей работой с другими людьми и постепенно формировать библиотеку часто используемых видов отображений. На этапе реализации описываемого эксперимента добавление данной возможности было отложено.

**3.8.2 Организация параметров материала**

Необходимо было обозначить способ определения набора параметров материала. Так как отображение шейдеров задается с помощью шейдерных программ, логично было предположить, что параметрами материала должны выступать переменные, используемые в шейдерах, если быть точнее, uniform-переменные. Uniform-переменные специально предназначены для передачи в шейдеры данных, которые одинаковы для всей расчётной сетки, и применяются при вычислении позиции каждого узла или цвета каждого пикселя.

Для того, чтобы определить, какие параметры включены в шейдер, необходимо провести простой синтаксический анализ исходного кода. Известно, что объявления всех uniform-переменных начинаются с ключевого слова «uniform», затем следует тип данных, затем имя переменной. Строчка заканчивается точкой с запятой.

Ниже приведен пример нескольких uniform-переменных, определенных в одном из шейдеров:

uniform mat4 view;

uniform mat4 projection;

uniform mat4 model;

uniform float angleX;

uniform float angleY;

uniform float angleZ;

uniform vec4 time;

Среди типов данных переменных можно заметить матрицы, векторы и числа с плавающей запятой. В программе также реализована поддержка всех типов данных, присутствующих в шейдерном языке программирование GLSL. Стоит отдельно обратить внимание на такие переменные, как «view», «projection», «model» и «time». Это предопределенные переменные, которые не отображаются в пользовательском интерфейсе и не могут быть настроены пользователем, а непосредственно вычисляются программой и передаются в шейдер. Например, переменные «projection», «model» и «view» — матрицы проекции, модели и вида, которые меняются в зависимости от расположения и характеристик камеры [11]. Это сделано с расчетом на то, что существует набор переменных, которые используются очень часто и нужны в широком наборе алгоритмов, при этом требуются особые вычисления для получения значений этих переменных.

**3.8.3 Настройка параметров материалов в пользовательском интерфейсе**

Описанные ранее возможности библиотеки Dear ImGUI были полезны для реализации настройки параметров в интерфейсе.

На рисунке 3.4 можно увидеть параметры, отображенные в пользовательском интерфейсе на основе приведенного в предыдущем подразделе исходного кода.

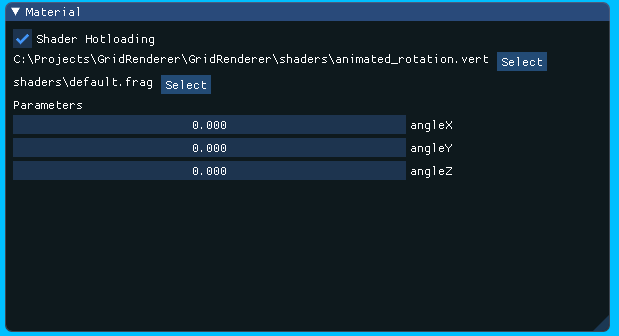


Рисунок 3.4 — пример отображения параметров материала в пользовательском интерфейсе

Как и было описано, переменные «view», «projection», «model» и «time» не отображаются в интерфейсе, так как это заранее определенные переменные, значение которых вычисляется динамически.

В пользовательском интерфейсе поддерживается отображение всех типов данных, присутствующих в языке GLSL, включая:

1. базовые численные типы, включая целые числа и числа с плавающей запятой разной точности;

2. векторы размерностей 2, 3 и 4;

3. матрицы с размерностями 2, 3 и 4 в различных комбинациях;

4. логический тип.

Также на рисунке 4.3 можно увидеть конфигурацию путей к файлам с исходным кодом вершинного и пиксельного шейдеров.

**3.8.4 Пример материала**

Для демонстрации работы описанной выше схемы далее будет приведен пример одного из тестовых материалов.

Исходный код вершинного шейдера:

#version 450 core

layout(location = 0) in vec4 inPosition;

layout(location = 1) in float inParam1;

layout(location = 2) in float inParam2;

layout(location = 3) in float inParam3;

out float param1;

out float param2;

out float param3;

uniform mat4 view;

uniform mat4 projection;

uniform mat4 model;

uniform vec3 offset;

void main(void)

{

param1 = inParam1;

param2 = inParam2;

param3 = inParam3;

gl\_PointSize = 2.0f;

gl\_Position = projection \* view \* model \* (inPosition + vec4(offset, 0.0));

}

Вершинный шейдер выполняет стандартные действия, такие как передача данных о расчетной сетке пиксельному шейдеру и базовый расчет позиции конкретного узла сетки в экранном пространстве. Дополнительно шейдер предоставляет возможность настроить смещение сетки относительно начала координат через интерфейс с помощью переменной «offset».

Исходный код пиксельного шейдера:

#version 450 core

in float param1;

in float param2;

in float param3;

out vec4 outColor;

uniform vec3 color;

uniform float threshold;

void main(void)

{

if (param1 > threshold)

discard;

outColor = vec4(color.r, color.g, color.b, 1.0);

}

Пиксельный шейдер вычисляет конечный цвет пикселя, в котором будет отрисован конкретный узел. Также шейдер позволяет настроить цвет сетки с помощью переменной «color» и порог «threshold», сравниваемый с одним из параметров, поставляемых в данных вместе с расчетной сеткой, в зависимости от которого узел может быть отсечен из отрисовки.

Сравнительный результат отображения сетки с помощью стандартных и приведенных выше шейдеров можно увидеть на рисунке 3.5. Вторая сетка смещена относительно первой и окрашена в зеленый цвет.

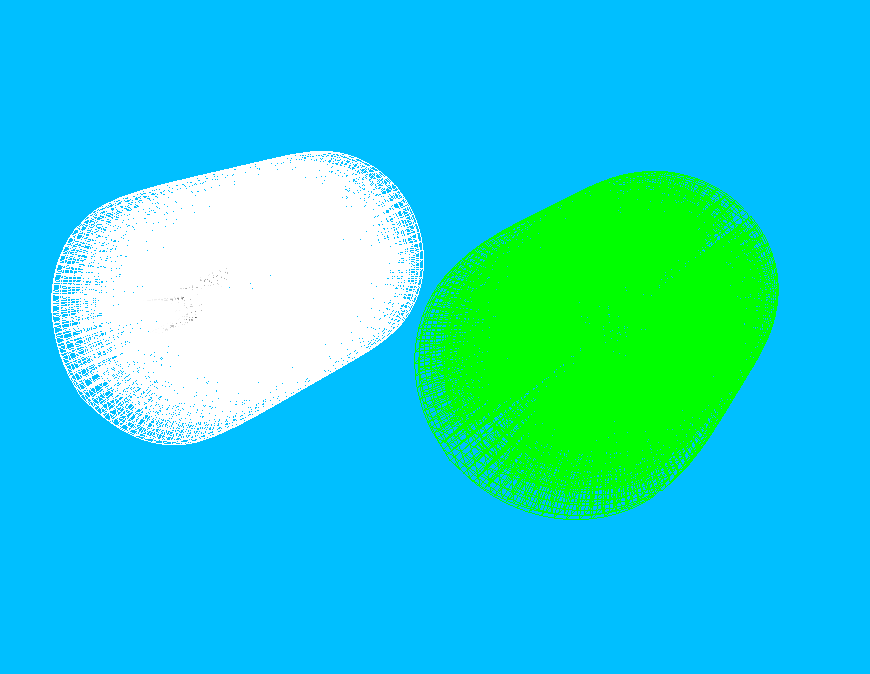


Рисунок 3.5 — сравнение сетки со стандартным материалом и сетки с отдельно созданным материалов

**3.9 Визуальное создание материалов**

**3.9.1 Концепция редактора узлов**

Описанная концепция материалов и её реализация показали, что такой подход обладает определенным потенциалом для решения поставленных задач. При этом, существует недостаток, мешающий полной реализации этого потенциала.

Схема, которая требует ручное написание шейдеров, подразумевает наличие у пользователя базовых знаний понятий программирования, а также требует изучение концепций программирования шейдеров и языка GLSL. Это недостаточно дружелюбный к пользователю подход, который может привести к возникновению множества мелких ошибок, что замедлит работу и не позволит сфокусироваться на цели исследователя, то есть изучении расчётных сеток.

Одним из возможных путей могло быть развитие синтаксического анализа написанных шейдеров и создание расширенного диалекта языка GLSL, специфичного для данной системы. Но это не решило бы обозначенную проблему. Во-первых, дополнительные синтаксические конструкции не упрощают изучение работы с программой. Во-вторых, расширение синтаксиса шейдерного языка делает разрабатываемый пакет несовместимым с другими программами.

Необходим способ дальнейшего упрощения создания новых визуальных отображений. Это приводит к выводу, что нужен некоторый визуальный способ задания алгоритмов для материалов, то есть некоторый аналог визуального программирования.

Если обратиться к популярным программам для 3D-моделирования, в них чаще всего можно увидеть редактирование материалов, основанное на создании графов, состоящих из узлов и связей между ними. Пример редактора материала в программе Blender [5] можно увидеть на рисунке 3.6.

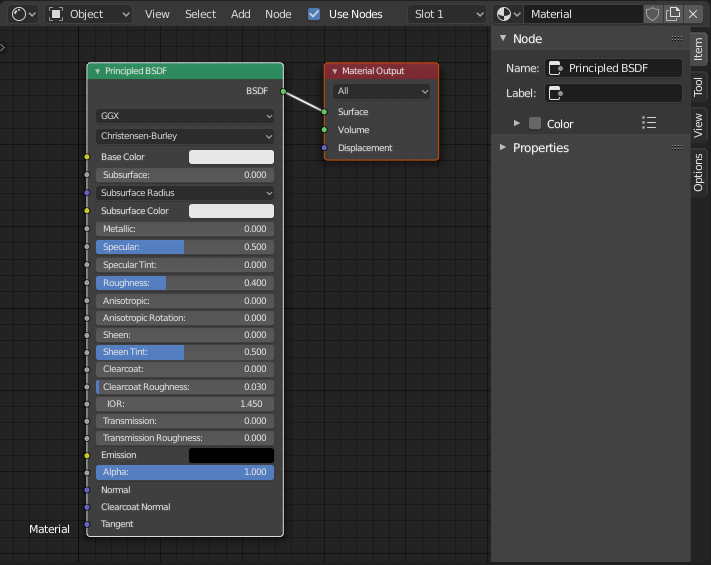


Рисунок 3.6 — визуальный редактор материалов в программе Blender

**3.9.2 Описание редактора узлов**

Как было ранее сказано, редактор узлов включает в себя два типа сущностей. Это узлы и связи.

Узел представляет собой прямоугольный блок. Узел обладает типом, определяющим функциональное предназначение узла и его визуальное отображение.

Есть несколько типов узлов:

1. uniform-узлы — предназначены для настройки uniform-переменных, описанных в подразделе 3.8.2;
2. узлы-атрибуты — предназначены для получения данных, хранящихся в атрибутах, описанных в подразделе 3.5;
3. узлы-операции — предназначены для преобразования входных данных и получения выходных;
4. выходные узлы — предназначены для получения итогового результата работы материала.

У каждого узла может быть несколько входов и выходов. У входных uniform-узлов и узлов-атрибутов есть только выходы. У выходных узлов есть только входы. У узлов-операций может быть произвольное число входов и выходов. У каждого входа и выхода есть тип данных. Набор типов данных ограничен типами данных, доступных в языке GLSL.

Узлы соединяются между собой с помощью связей через входы и выходы. Типы данных соединенных входа и выходы должны совпадать. Связи между узлами указывают на направление движения данных. При создании связей следует избегать появления циклов, так как такие ситуации не имеют смысла в рассматриваемой концепции, о чем будет сказано чуть более подробно позже в описании технической реализации. Говоря формально, корректный материал представляет собой ациклический направленный граф.

У пользователя есть возможность определять собственные узлы с помощью создания файлов, содержащих конфигурацию нового узла. Об этом написано более подробно далее.

Пример материала, настроенного в редакторе узлов, можно увидеть на рисунке 3.7. Это простейший материал, который совершает преобразование некоторого настроенного цвета в вектор, который затем используется в качестве результата работы материала. Код, сгенерированный для материала, можно увидеть в приложении А.

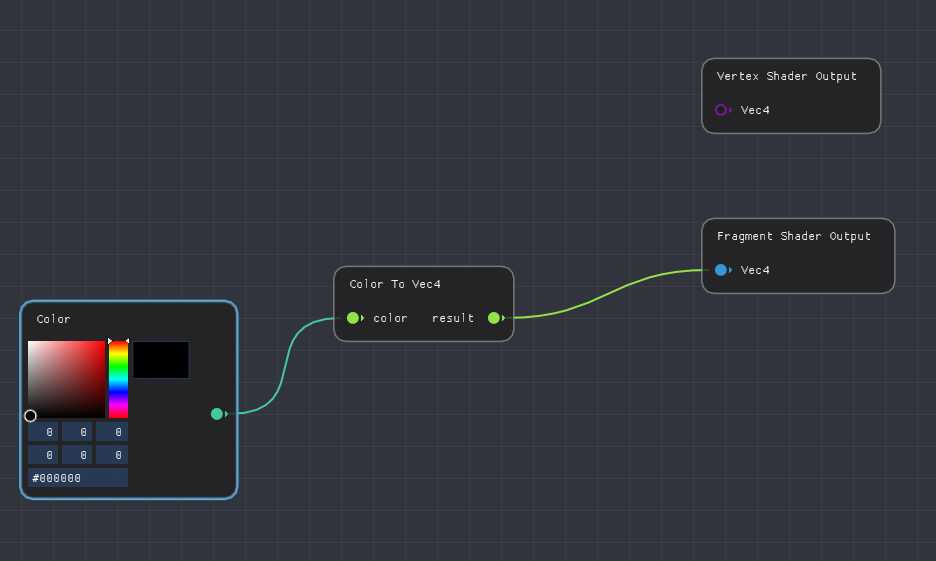


Рисунок 3.7 — пример простейшего материала, созданного в редакторе узлов

**3.9.3 Генерация шейдеров**

Графы материалов, настроенные в редакторе узлов, преобразуются в шейдеры. Таким образом достигается отображение материала. Параметры, настраиваемые через входные узлы, в реальном времени передаются на графический ускоритель.

Основная техническая сложность заключается в процессе преобразования графа в исходный код шейдера. Остальные элементы системы в целом уже были реализованы на предыдущих итерациях.

Процесс генерации исходного кода шейдера выглядит следующим образом:

1. находится выходной узел материала;
2. начинается рекурсивный обход ациклического направленного графа в глубину;
3. если посещаемый узел — узел-операция, то в главную функцию шейдера добавляется новая часть кода в соответствии с типом операции, про то, как обрабатываются узлы операции, можно подробнее прочитать в следующем подразделе;
4. если посещаемый узел — uniform-узел или узел-атрибут, то в код шейдера добавляется объявление uniform-переменной либо атрибута соответственно;
5. при возвращении из рекурсии для каждого узла-операции подставляются переменные в код операции в нужных местах исходя из того, с какими узлами связаны входы данного узла;
6. на последнем этапе отдельные секции кода шейдера, такие как объявления атрибутов, объявления uniform-переменных, главная функция шейдера, объединяются в единый шейдер, добавляются необходимые технические спецификации.

**3.9.4 Библиотека узлов-операций**

Каждый тип узла-операции представляет собой некоторое преобразование данных. Важный момент для обеспечения расширяемости — возможность добавлять новые виды преобразований.

Все узлы-операции организованы в библиотеку текстовых файлов, где каждый файл определяет один узел-операцию. Добавление, удаление и изменение текстовых файлов — понятный для каждого пользователя компьютера процесс. Некоторый набор стандартных узлов-операций поставляется вместе с программой, пользователи вольны добавлять собственные и делиться ими друг с другом.

Файл любого корректного узла-операции содержит описание входов, выходов узла и выполняемый код, подставляемый в исходный код шейдера. Пример описания простого узла, выполняющего умножения вектора на число:

Multiply Vec4

in vec4 vector

in float value

out vec4 result

vec4 {2} = {0} \* {1};

Первая строчка содержит название операции, которое видит пользователь при выборе операции из меню. Далее несколько строчек определяют входы и выходы узла. Сначала идут входы, обозначаемые ключевым словом «in» в начале строки, затем выходы, обозначаемые ключевым словом «out». Для каждого входа и выхода необходимо указать тип данных и осмысленное имя. Остаток файла содержит исходный код, который будет подставлен в исходный код шейдера. Он может занимать произвольное количество строчек. В местах кода, где должны быть подставлены определенные ранее входные и выходные переменные, указываются числа, указывающие на индекс переменной в списке, заключенные в фигурные скобки.

**3.9.4 Сохранение и загрузка материалов**

Для сохранения и загрузки материалов была подобрана специализированная библиотека для языка C++ под названием ajson. С её помощью хранение структур данных языка можно настраивать следующим образом:

struct Person

{

std::string Name;

int Age;

};

AJSON(Person , Name , Age)

Person obj;

char \* json= "{ \"Name\" : \"Boo\", \"Age\" : 28}";

ajson::load\_from\_buff(obj,json);

Каждый материал хранится в отдельном файле в формате JSON.

**3.9.5 Организация сцены**

Организация сцены в проекте позволяет расположить в пространстве рядом друг с другом несколько сеток. У каждой сетки можно настраивать позицию поворот и масштаб, а также для каждой из них можно назначить и настроить свой материал. На рисунке 3.8 можно увидеть настройки сцены в пользовательском интерфейсе. Здесь представлен список загруженных сеток, настройки их позиций, поворотов, масштабов, выбор режима представления и настройка видимых частей сетки.



Рисунок 3.8 — настройки сцены

**3.10 Результаты и их обсуждение**

**3.10.1 Примеры отображений**

Ниже можно увидеть пример отображения сетки с помощью графа, показанного на рисунке 3.9.

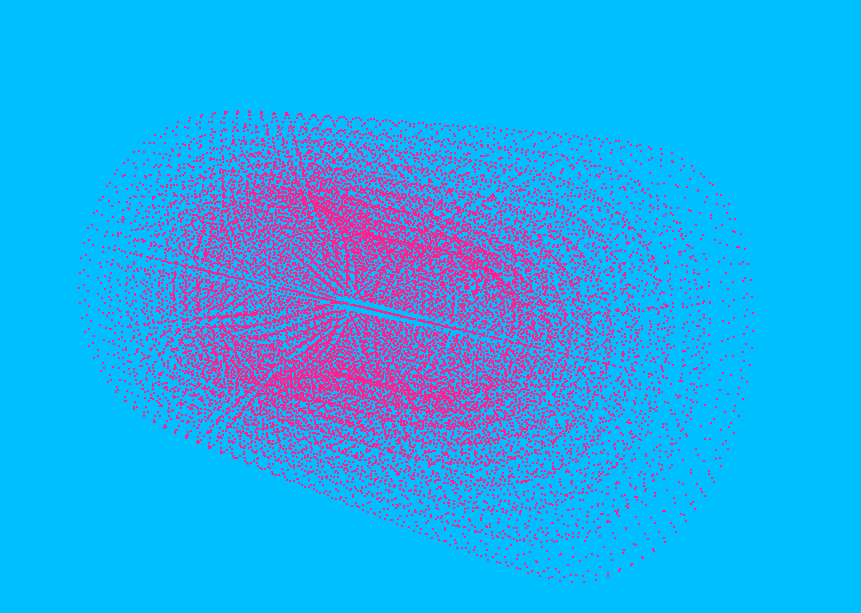


Рисунок 3.9 — простейшее отображение сетки одним цветом

Это минимальный пример, который позволяет просто отобразить данные каким-то одним настраиваемым цветом.

Пример отображения некоторого параметра, заданного для каждого узла сетки можно увидеть на рисунке 3.10. На рисунке 3.11 проиллюстрирован граф в редакторе узлов.

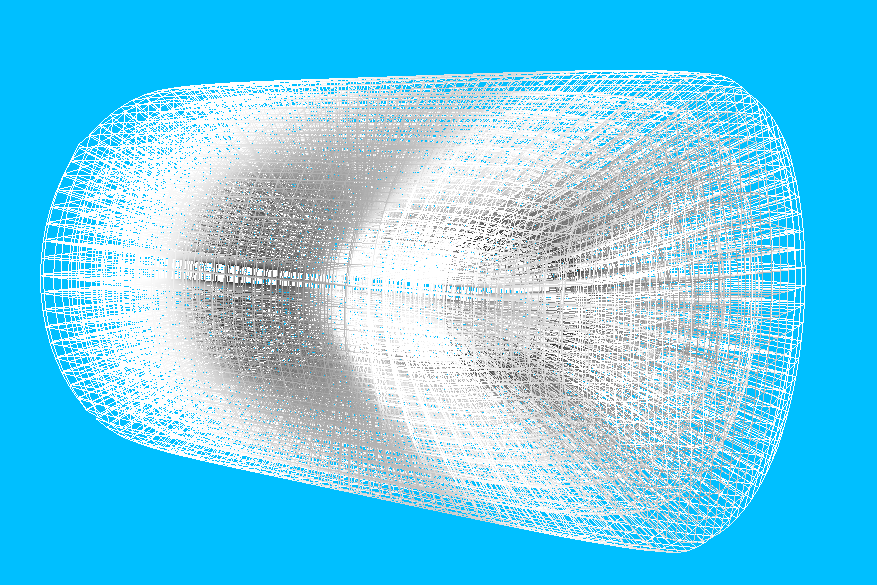


Рисунок 3.10 — отображение с параметром

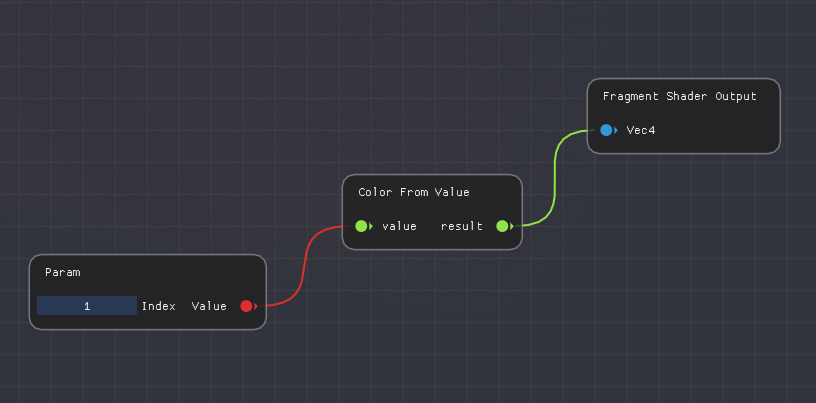


Рисунок 3.11 — настройка отображения с параметром

Если поменять отображаемый параметр, соответственно меняется отображение, как видно на рисунке 3.12 (видно небольшие вкрапления темных оттенков серого цвета внутри сетки).

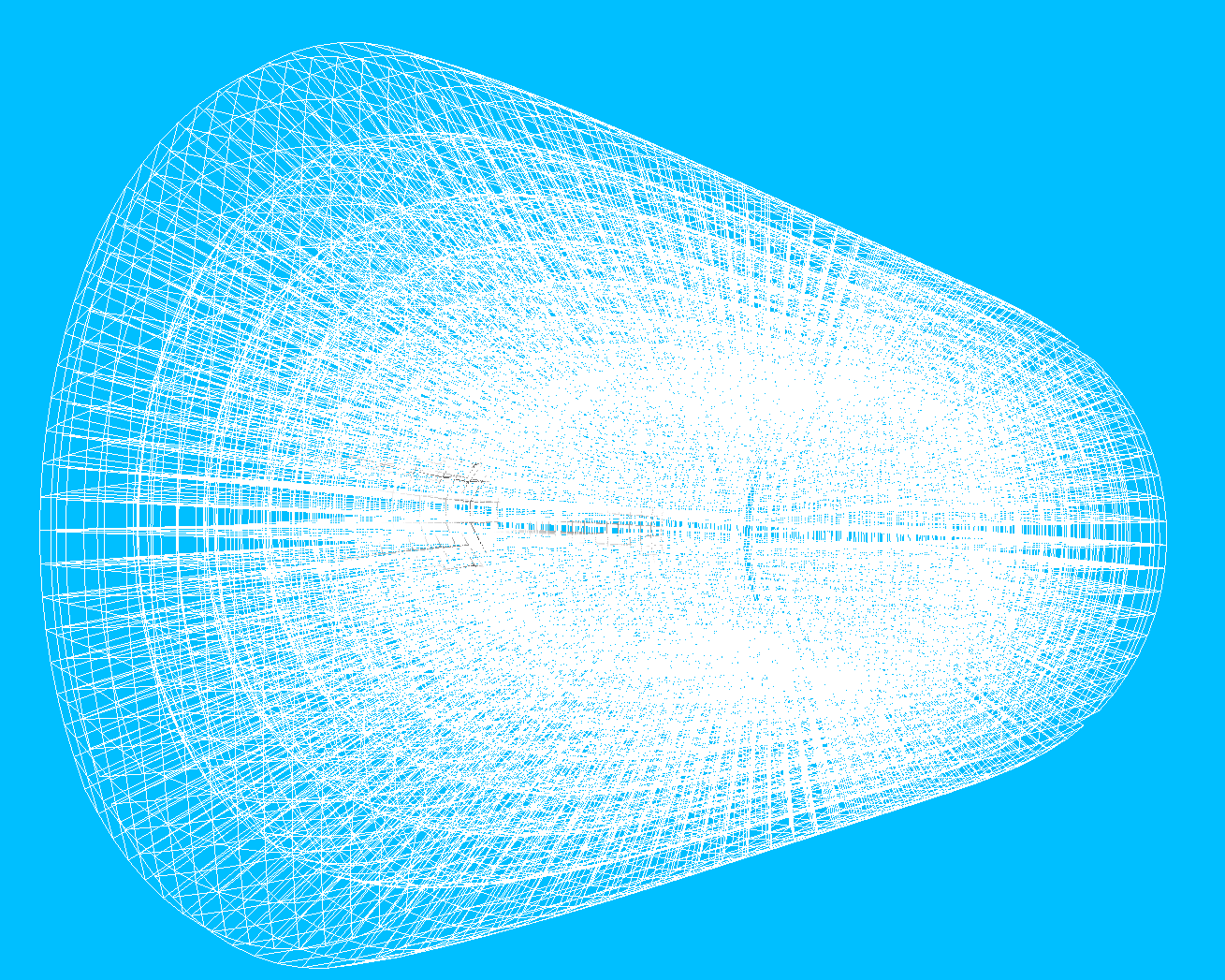


Рисунок 3.12 — отображение с другим параметром

Объединив отображения данных цветом и параметром, как показано на рисунке 3.14, мы получим результат, который видно на рисунке 3.13. Как видно, комбинирование различных операций происходит посредством соединения заранее созданных отображений с помощью некоторого объединяющего узла.

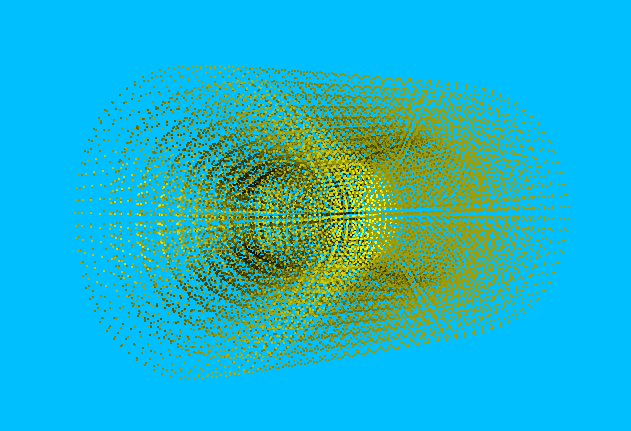


Рисунок 3.13 — отображение с параметром и цветом

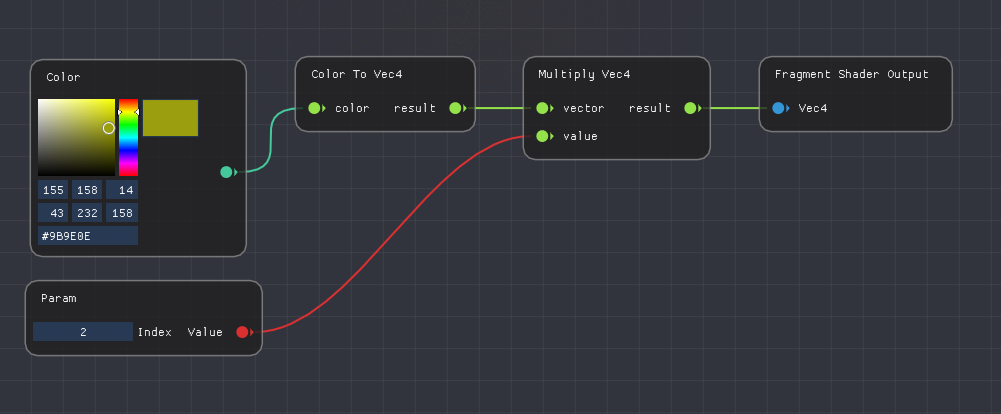


Рисунок 3.14 — настройка отображения с параметром и цветом

С помощью условных узлов мы можем некоторым образом фильтровать данные, манипулирую значениями индексов узлов. Пример показан на рисунках 3.15 и 3.16. Если граф становится громоздким, можно выделить часто используемые наборы операций в отдельные узлы.

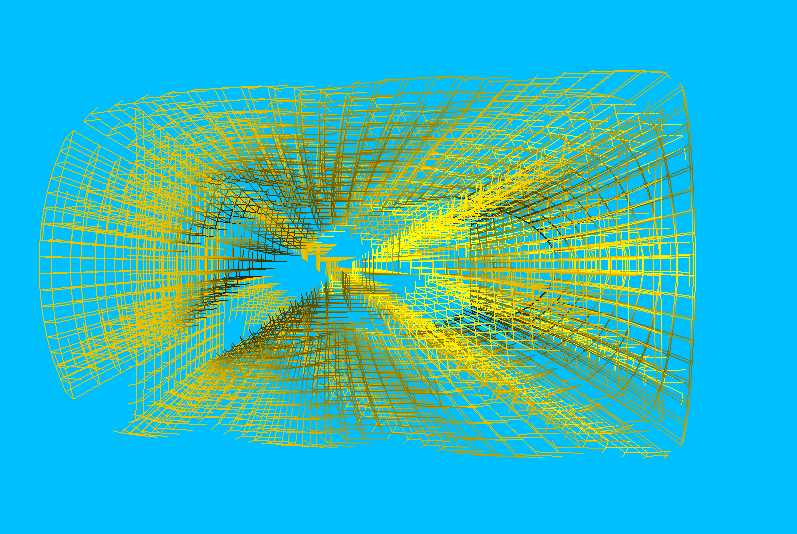


Рисунок 3.15 — простейшая фильтрация сетки по индексам узлов

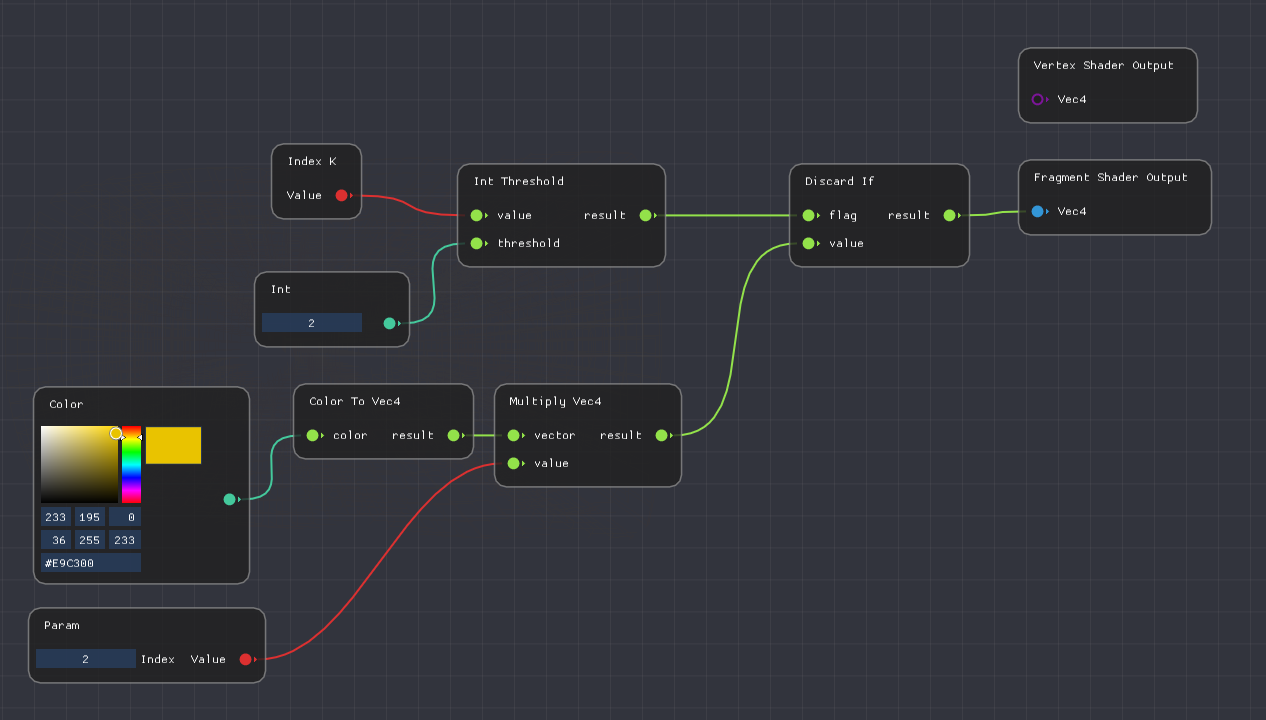


Рисунок 3.16 — настройка фильтрации сетки по индексам узлов

Один и тот же материал можно применять для разных наборов данных, что видно на рисунке 3.17.

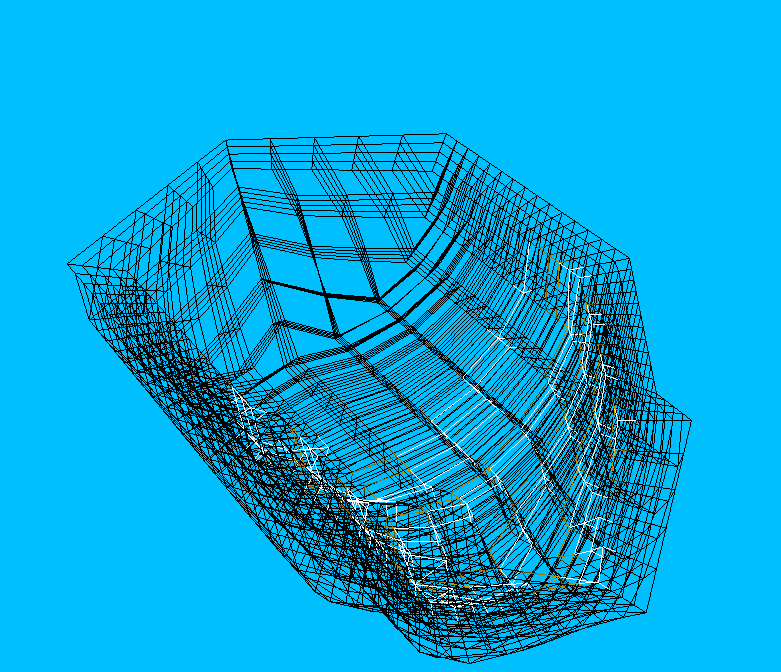


Рисунок 3.17 — материал, примененный для другой сетки

**3.10.3 Направления доработок и улучшений**

Среди направлений дальнейшей работы можно выделить следующие пункты:

1. приведения типов. На момент написания текста узлы могут быть соединены, только если у входа и выходы один и тот же тип данных. Это приводит необходимости определять дополнительные узлы-операции, предназначенные для преобразования типов данных. Этого можно избежать, если автоматически приводить тип если это возможно. Например, целое число к числу с плавающей запятой;
2. полноценная поддержка вершинных шейдеров. Это позволит свободно манипулировать положением узлов, ячеек сеток в той же манере, как это сейчас реализовано с цветами;
3. поддержка геометрических шейдеров. Геометрические шейдеры позволяют генерировать дополнительную геометрию на этапе выполнения шейдера, что значительно расширяет набор возможных функций, которые можно реализовать;
4. улучшенная библиотека узлов-операций. Было бы удобно распределять узлы по категориям, располагая файлы в иерархии папок;
5. вложенные графы узлов. Введение нового уровня абстракции позволит повысить возможность повторного использования готовых графов узлов, выполняющих часто функциональность;
6. внедрение виртуальной реальности. Данное направление можно назвать отдельным и не относящимся напрямую к описанной концепции редактора узлов. Тем не менее, в долгосрочной перспективе кажется интересной попытка реализации создания новых визуальных отображений в виртуальной трехмерной реальности, основываясь на идеях, используемых в двумерном редакторе узлов.

# ЗАКЛЮЧЕНИЕ

В результате проведенной работы был разработан специализированный графический пакет для визуализации расчётных сеток. Внедрение редактора узлов является логичным продолжением предшествующих экспериментов, направленных на обеспечение гибкости и расширяемости функциональности системы со стороны пользователя.

Реализован алгоритм генерации исходного кода шейдеров на основе направленных ациклических графов, настраиваемых визуально. Система не испытывает проблем с производительностью на заявленных объемах данных. Потребность в оптимизации может возникнуть только при создании действительно сложных и нетривиальных визуальных отображений на большом количестве данных.

Дальнейшее развитие данного направления перспективно, адаптация дополнительных функций из популярных программных пакетов для 3D-моделирования с учетом потребностей научной визуализации может принести значительную пользу в долгосрочной перспективе.

Возможно комбинирование данного подхода с различными смежными направлениями, такими как виртуальная реальность, управление жестами, онлайн-визуализация и другими.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ

1. Анненкова О.Г., Специализированная система визуализации сеточных данных: магистерская дис. — Екатеринбург, 2016. — 42 с.
2. Васёв П.А., Бахтерев М.О., Декларативно-императивный метод конструирования сцен научной визуализации // XV Международная конференция «Супервычисления и Математическое Моделирование». Тезисы. ФГУП «РФЯЦ ВНИИЭФ». Саров. 2014, 46 с.
3. Ajson [Electronic resource] — Mode of access: https://github.com/lordoffox/ajson.
4. Bailey M., Cunningham S., Graphics Shaders. Theory and Practice. Second Edition // CRC Press, New York, 2012.
5. Blender [Electronic resource] — Mode of access: https://blender.org.
6. Dear ImGUI [Electronic resource] — Mode of access: https://github.com/ocornut/imgui.
7. Fmt [Electronic resource] — Mode of access: https://github.com/fmtlib/fmt.
8. Git [Electronic resource] — Mode of access: https://git-scm.com.
9. Imgui-node-editor [Electronic resource] — Mode of access: https://github.com/thedmd/imgui-node-editor.
10. Meiri E. OpenGL Step by Step [Electronic resource] / Etay Meiri — Mode of access: http://ogldev.atspace.co.uk/index.htm.
11. Lengyel E., Foundations of Game Engine Development Volume 1: Mathematics // Terathon Software LLC, Lincoln, 2016.
12. Lengyel E., Mathematics for 3D Game Programming and Computer Graphics // Course Technology PTR, Boston, 2012.
13. Olsannikova E., Ometov A., Koucheryavy Y. and Olsson T., Visualizing Big Data with augmented and virtual reality: challenges and research agenda. // Journal Of Big Data, 2015, pp.1-27.
14. ParaView [Electronic resource] — Mode of access: https://paraview.org.
15. SDL Wiki [Electronic resource] — Mode of access: https://wiki.libsdl.org.
16. Shreiner D., OpenGL Programming Guide // Addison-Wesley, Boston, 2013.
17. Stroustrup B., The C++ Programming Language // Addison-Wesley, Boston, 2013.
18. Tecplot [Electronic resource] — Mode of access: https://tecplot.com.
19. Ushakova O.V., Advances in Grid Generation // Nova Science Publishers, New-York, 2007.
20. Visual Studio [Electronic resource] — Mode of access: https://visualstudio.microsoft.com.
21. Wright R.S., Haemel N., Sellers G., Lipchak B., OpenGL SuperBible // Addison-Wesley, Boston, 2010.

# ПРИЛОЖЕНИЕ А

Ниже можно прочитать код вершинного шейдера, сгенерированного для материала, представленного на рисунке 4.6:

#version 450 core

layout(location = 0) in vec4 inPosition;

layout(location = 1) in float inParam1;

layout(location = 2) in float inParam2;

layout(location = 3) in float inParam3;

layout(location = 4) in int inIndexI;

layout(location = 5) in int inIndexJ;

layout(location = 6) in int inIndexK;

out vec4 outPosition;

out float param1;

out float param2;

out float param3;

flat out int indexI;

flat out int indexJ;

flat out int indexK;

uniform mat4 view;

uniform mat4 projection;

uniform mat4 model;

void main(void)

{

outPosition = inPosition;

param1 = inParam1;

param2 = inParam2;

param3 = inParam3;

indexI = inIndexI;

indexJ = inIndexJ;

indexK = inIndexK;

gl\_Position = projection \* view \* model \* inPosition;

gl\_PointSize = 2.0f;

}

Код пиксельного шейдера:

#version 450 core

in vec4 position;

in float param1;

in float param2;

in float param3;

flat in int indexI;

flat in int indexJ;

flat in int indexK;

uniform vec4 var\_7;

out vec4 outColor;

void main(void)

{

outColor = var\_7;

}

# ПРИЛОЖЕНИЕ Б

Репозиторий с исходным кодом проекта находится в открытом доступе на сайте Github по ссылке https://github.com/catinjar/GridRenderer.

Для открытия и компиляции проекта требуется наличие Visual Studio 2019 на компьютере разработчика. При распространении программы требуется только установка актуального распространяемого пакета Microsoft.