МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «КубГУ»)

Факультет математики и компьютерных наук

Кафедра математических и компьютерных методов

Допустить к защите
Заведующий кафедрой
канд. физ.-мат. наук, доцент
_____ А. В. Лежнев
_____ 2020 г.

Руководитель ООП
докт. физ.-мат. наук, проф.
_____ В. Г. Лежнев
_____ 2020 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ)

АЛГОРИТМЫ РАСЧЕТА ОБЪЕМА ТЕХНИЧЕСКОГО СЫРЬЯ ЛАЗЕРНЫМ ИЗМЕРИТЕЛЬНЫМ КОМПЛЕКСОМ ДЛЯ WEB-ПРИЛОЖЕНИЙ

Работу выполнил _____ Н. А. Барышников
(подпись)

Направление подготовки 02.04.01 Математика и компьютерные науки

Направленность (профиль) Математические методы теории сложных систем

Научный руководитель
д.ф.-м.н., доцент,
проф.каф.МКМ_ _____ С. В. Усатиков
(подпись)

Нормоконтролер
ст. лаборант _____ И. С. Пшикова
(подпись)

Краснодар
2020

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Точные замеры объемов сыпучих материалов – это неотъемлемая часть работы горнорудных комбинатов, шахт, нефтеперерабатывающих заводов, химических производств, различных транспортных компаний и предприятий агропромышленного комплекса. Это вызвано тем, что сыпучие материалы сложно поддаются точному учету.

При этом сыпучее сырьё составляет значительную часть расходов предприятий и для эффективного его распределения необходимо контролировать объём.

Необходимая точность определения объемов в большей степени зависит от стоимости единицы объема материала и может варьироваться от единиц до долей процентов от общего объема материалов. Чем дороже материал, тем с большей точностью требуется вычислить его количество. Высокие требования к точному подсчету земляных работ предъявляют в настоящее время и строители. Объемы перемещённого грунта на больших строительных площадках измеряются миллионами кубометров, и ошибка в определении реальных трудозатрат может стоить миллионы рублей.

Традиционно используются два метода измерения объёма неровных тел: вычисление объёма тела при помощи ёмкости с водой и вычисление объёма тела через разбиение его на правильные геометрические фигуры. Первый метод очень сложно организовать с габаритными объектами. Реализация второго представляет собой сложную задачу, если нельзя явно выделить правильные геометрические фигуры. В данной работе предлагается алгоритм вычисления объёма габаритного объекта с неровной поверхностью, заданной дискретными значениями.

Цель работы – разработка алгоритма расчета объема технического сырья, где благодаря лазерному измерительному комплексу известна линия поверхности насыпного материала.

Задачи:

- изучение действующих измерительных комплексов, и определение требований к нему, с получением многомерных распределений вероятностей форм и размеров сырья;

- разработка алгоритма расчета объема сырья и анализ сопутствующих факторов;

- реализация программного обеспечения с помощью объектно-ориентированного программирования на платформе asp.net core и web разработки.

1 Математическое обеспечение действующих измерительных комплексов и возможностей расширения их функциональности, с применением детерминированных и стохастических моделей и методов

1.1 Приборы, использующиеся при измерении объема технического сырья

1.1.1 Основные комплексы и их разновидности

В разных областях человеческой деятельности возникает задача исследования параметров габаритного объекта, в частности измерения объёмов сыпучих тел. Речь, прежде всего, идет об учёте таких материалов, как: сельскохозяйственная продукция (зерно, клубни, подсолнечник), стройматериалы (песок, гравий, щебень, цемент), полезные ископаемые (уголь, руда).

Необходимая точность определения объёмов [19] в существенной степени зависит от стоимости единицы объёма данного материала или товара и может варьироваться от единиц вплоть до долей процентов от общего объёма материалов. Чем дороже материал, тем с большей точностью требуется вычислить его количество. Высокие требования к точному подсчету земляных работ предъявляют в настоящее время и строители. Объёмы перемещённого грунта на больших строительных площадках измеряются миллионами кубометров, и ошибка в определении реальных трудозатрат может стоить миллионы рублей. Требования к точности определения объёма часто могут достигать 1–3% от общего объёма склада. И сделать это очень непросто.

Задача точного измерения объёмов сыпучих материалов имеет широкое применение в следующих областях:

- в строительстве (определение объёмов выемки грунта и объёмов насыпей);
- в горнорудной промышленности (измерение объёмов выработки);

- при инвентаризации складов сырья и готовой продукции (уголь, песок, щебень, руда и рудные концентраты, зерно, клубни и прочее).

Существующие приборы и инструменты для измерения сырья:

1. Тахеометр. Благодаря этому прибору можно получить особенно точные расчеты с минимальной погрешностью. Однако для выполнения исчислений приходится тратить много времени. Изображен на рисунке 1.

2. Лазерный дальномер. Улавливает отраженный от препятствия лазерный луч и вычисляет расстояние по сдвигу фазы сигнала, которым этот луч модулируется. Лазерные дальномеры отличаются высокой точностью измерений – до десятых долей миллиметра.

3. 3*D* лазерный сканер. Производит до миллиона измерений в секунду, представляет объекты в виде набора точек с пространственными координатами. Полученный массив данных, называемый облаком точек, может быть впоследствии представлен в трехмерном и двухмерном виде, а также использован для измерений, расчетов, анализа и моделирования.

4. Оптическое оборудование. Здесь чаще всего речь идет о беспилотных летательных аппаратах, способных выдавать неплохие результаты. Однако эти приборы обладают высокой стоимостью.

5. Приборы, основанные на технологии низкочастотных звуковых импульсов. Использует низкочастотный акустический сигнал, который легко распространяется при наличии пыли, и обеспечивает измерение расстояния, основываясь на информации о времени прямого и обратного прохождения сигнала до поверхности продукта.

6. Волноводный радар. Менее чувствителен к неровности поверхности, поскольку длина микроволнового сигнала меньше, и его направление задается зондом. Бесконтактные радары попадают под влияние неровных поверхностей. Для компенсации устройство собирает небольшие эхосигналы и объединяет их в один эхосигнал.

Традиционным инструментом [19] для определения объёмов сыпучих материалов и грунта как при инвентаризации складов, так и при земляных

работах является обыкновенная геодезическая съёмка (обычно с помощью электронного тахеометра), позволяющая определить объёмы материалов с точностью не выше 5–10%. Основным ограничением при проведении работ с применением тахеометров является низкая скорость измерений и физическая невозможность детальной съёмки больших объёмов материалов.

Рисунок 1 – Прибор тахеометр

В электронно-оптических тахеометрах [19] расстояния измеряются по разности фаз испускаемого и отражённого луча (фазовый метод), а иногда (в некоторых современных моделях) – по времени прохождения луча лазера до отражателя и обратно (импульсный метод). Точность измерения зависит от технических возможностей модели тахеометра, а также от многих внешних параметров: температуры, давления, влажности и тому подобное.

В отличие от замеров объёмов хранения сыпучих материалов посредством тахеометров или лазерных дальномеров, технология 3*D* лазерного сканирования [19] позволяет детально, с шагом до единиц миллиметров, обмерить и отразить форму бурта или поверхности кучи материала на складе. Такая детальность обмеров лазерным сканером позволяет получить точность обмеров, не достижимую при применении любой другой существующей технологии замера объёма сыпучих материалов. Высокая скорость работы сканера, составляющая десятки и сотни тысяч измерений в секунду, и современное программное обеспечение для обработки полученных результатов наземного лазерного сканирования позволяют в

сжатые сроки провести обмеры складов сыпучих материалов и вычислить объёмы с погрешностью до 1%.

1.1.2 Лазерные измерительные приборы для ленточного конвейера

Установка по измерению объема сыпучих материалов предназначена для бесконтактного измерения объема сыпучих материалов, транспортируемых ленточным конвейером (рисунок 2). Также может использоваться для определения количества перемещаемых по конвейеру предметов. Результаты измерения отображаются на экране ПК (персональный компьютер) или на индикаторе аналогового устройства. Эта установка незаменима в производстве строительных материалов, добывающей, химической, целлюлозно-бумажной, пищевой промышленности, обработки сельхозпродукции.

Установка выполняет следующие функции:

- проведение бесконтактных измерений уровня, объема сыпучего сырья, материалов и заготовок в процессе производства в реальном времени;
- расчет и индикация объёма материала за заданный промежуток времени;
- возможность удаленного управления установкой посредством сети *Ethernet*;
- отображение измеренного профиля сыпучих материалов в заданном сечении на экране ПК;
- сохранение результатов измерения на ПК с возможностью их дальнейшего просмотра в графическом и табличном виде;
- создание, изменение и хранение базы данных результатов измерений;
- возможность вывода на печать результатов измерений.

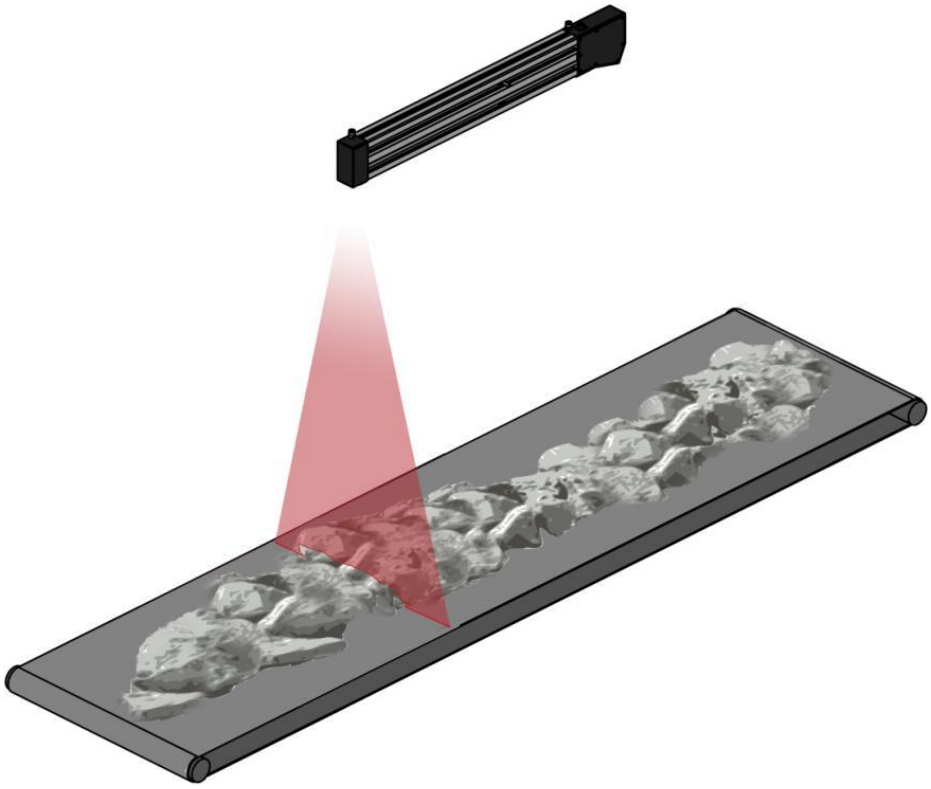

Рисунок 2 – Устройство установки

Для измерения объема грузопотока над полотном конвейера размещается лазерный сканер. Направление сканирования перпендикулярно направлению движения ленты конвейера, что можно увидеть на рисунке 3.

В начале работы формируется и задаётся нулевой профиль конвейерной ленты путём её сканирования без материала.

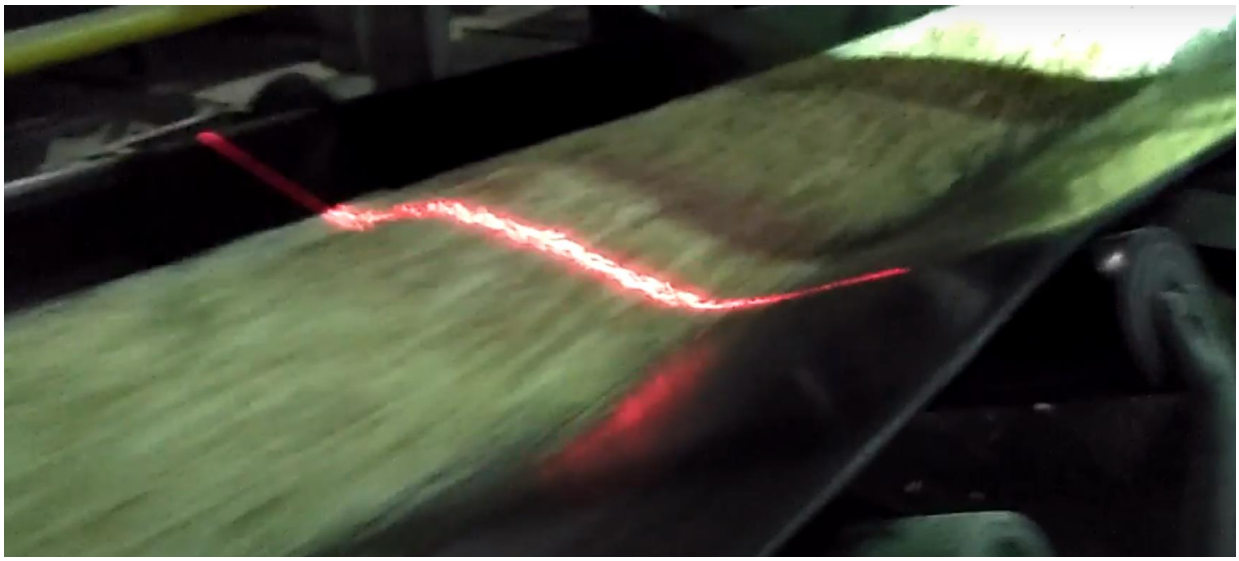

Рисунок 3 – Фото на предприятии

Лазерный датчик сканирует профиль конвейерной ленты и передает результаты измерений на компьютер, в котором вычисляется объем грузопотока за заданный интервал времени. Применение алгоритмов фильтрации позволяет исключить ошибочные данные и повысить точность измерений. Программное обеспечение формирует отчет о результатах измерений [9].

1.2 Техническое сырье, добыча, использование, особенности

1.2.1 Щебень

Щебень является самым основным минеральным сырьем в мире. Он обилен, широко доступен и недорог. Это материал, с которым люди знакомы почти во всех частях мира.

Щебень [2], виды которого будут более подробно описаны ниже и которые изображены на рисунке 4, является строительным материалом, полученным в результате первоначального измельчения и последующего просеивания горных пород. Он входит в состав бетонной смеси для фундамента, и его характеристики во многом определяют прочность раствора. Особенно это касается фундаментов, на которые во время эксплуатации дома ложатся большие нагрузки. А долговечность всей конструкции будет зависеть от прочности фундамента жилого дома или здания другого назначения.

Классификация щебня[1].

Этот материал классифицируется по нескольким основным признакам. Среди них следует выделить: тип породы, прочность и морозостойкость. В порядке повышения прочности следует различать следующие виды щебня:

- шлаковый;
- вторичный;
- известняковый;
- гравийный;

- гранитный;
- сланцевый.

Рисунок 4 – Виды щебня

Самым прочным и надежным является гранитный, он выступает как оптимальный вариант для заливки фундамента. Но если учесть две характеристики: стоимость и долговечность, то гравийный щебень будет считается лучшим. Вторичный щебень получают путем дробления бетонных отходов, а также дробления кирпича. Прежде чем использовать этот материал, необходимо позаботиться для того, чтобы извлечь старую арматуру.

Большое значение для строительства в условиях сурового климата имеет количество циклов замораживания и оттаивания, которые смогут проходить щебень, не теряя своих качественных характеристик. Таким образом, с точки зрения морозостойкости, материал может относиться к марке в диапазоне от F15 до F400. Чаще всего строители обращают внимание на эти показатели, однако щебень можно классифицировать по некоторым вспомогательным характеристикам, например, по уровню адгезии или радиоактивности.

Характеристики и применение гранитного щебня[1].

Это неметаллический строительный материал, который получают из твердых горных пород. Мерзлая магма имеет вид монолитной породы, которая добывается со значительной глубины. При изготовлении этого материала используются ГОСТы 8267-93. Также он делится на фракции. Таким образом, размеры зерен в материале могут быть не менее 0-5 мм, а максимальные-от 150 до 300 мм.

Гранитный щебень считается наиболее распространенным среди потребителей, его фракция колеблется от 5 до 20 мм. Этот материал используется в производстве асфальта и бетона. Гранитный щебень применяется при закрытии раствора для образования железобетонных конструкций, железнодорожных путей, при закладке фундаментов автомобильных дорог, а также тротуаров и площадок.

Характеристики и применение шлакового щебня.

Данный вид щебня является доступным строительным материалом, получаемым из отходов металлургической и химической промышленности, а также при сжигании твердого топлива. Шлаковый щебень обладает высокой плотностью. Его плотность выше, чем у гранитного щебня. Однако, высокая плотность характеризует большую массу материала, что в строительстве является достаточно весомым недостатком. Шлаковый щебень имеет более высокий коэффициент водопоглощения, по сравнению с гранитным. Поэтому конструкции, возведенные с таким щебнем, не должны часто контактировать с влагой.

Шлаковый щебень также не должен подвергаться частому замораживанию, ведь характеристики по морозостойкости этого материала весьма малы (всего 15 циклов против 300 циклов гранитного щебня). Применение шлакового щебня целесообразно в случае полноценного учета всех его недостатков. В целом, в зависимости от фракции (0-5, 5-20, 20-40, 40-70, 70-120 мм) шлаковый щебень достаточно часто применяют в гражданском и дорожном строительстве, а также в производстве строительных материалов.

Характеристики и применение сланцевого щебня

Это материал, получаемый из горных пород вулканического происхождения. Внешне сланцевый щебень будет выглядеть, как насыпь камня продолговатой, плоской формы. По цвету сланцевый щебень может быть бордовым, желтым, коричневым, серым, антрацитовым или зеленым. Сланцы довольно широко применяют в строительстве и производстве материалов. Например, существует порода сланцев, при раскалывании которых, образуется множество тончайших пластинок. Такие сланцы применяют для кровельных материалов.

Продукт дробления сланца – щебень [3], также широко используется в строительстве. Сланцевый щебень применяют для строительства дорог и монолитных железобетонных конструкций, а также для декорирования ландшафта. В зависимости от фракции, сланцевым щебнем украшают элементы декора прилегающей территории, создают интересные дорожки.

Характеристики и применение гравийного щебня.

Этот вид щебня производится путем прохождения через специальное сито горного образования или дробления каменных пород. ГОСТ 8267-93 используется в качестве нормативного документа для производства продукции. Этот вид щебня уступает граниту по прочности. Среди преимуществ следует выделить низкую радиоактивность, а также низкую стоимость. Рассматривая виды щебня, необходимо различать его разновидности, среди которых стоит отметить дробленый и гравийный.

Первый производится методом обработки горной породы, в то время как второй представляет собой гальку речного и морского происхождения. Гравийный щебень применяется в качестве заполнителя при формировании изделий, а также железобетонных конструкций. Он применяется в строительстве, в процессе покрытия пешеходных дорог, а также при строительстве фундаментов и площадок.

Характеристики и применение известнякового щебня [2].

Рассматривая виды щебня и его применение, потребители выделяют известняковую разновидность, которая представляет собой материал,

полученный по технологии переработки осадочных пород. В качестве сырья используется известняк, который состоит из карбоната кальция и имеет низкую стоимость.

Известняковый щебень можно встретить в стекольной и полиграфической промышленности. Он активно используется при изготовлении мелких бетонных изделий, при строительстве дорог, на поверхности которых не будет большой транспортной нагрузки во время эксплуатации.

Характеристики и применение вторичного щебня.

Этот материал изготавливается по технологии переработки строительных отходов, а именно: асфальта, бетона и кирпича. Материал должен соответствовать ГОСТ 25137-82. Он использует ту же технологию, что и при изготовлении других видов щебня. Главным преимуществом является низкая цена. По характеристикам прочности и морозостойкости этот материал уступает природным сортам щебня. Он применяется в дорожном хозяйстве, в роли заполнителя для бетона, а также при укреплении слабых грунтов.

Кроме вышеперечисленных сортов, в последнее время выпускается вторичная крошка, которая является отходом производства, с использованием дробленого щебня и непригодных железобетонных изделий. Этот вид щебня является самым дешевым и используется для формирования верхнего слоя улиц зимой [1].

1.2.2 Песок

Песок – это осадочная порода и искусственный материал, который состоит из фракций горных пород (рисунок 5). Довольно часто он состоит из минерального кварца, который представляет собой вещество, называемое кремнеземом.

Это материал, образовавшийся в результате естественных процессов. По сути, это осадочная горная порода, разрушенная под воздействием солнца, ветра и осадков. Процесс разрушения может протекать столетиями.

Отделившиеся частицы породы могут перемещаться и оседать в разных местах нашей планеты: на суше и под водой.

В зависимости от этого, выделяют несколько типов отложений:

- аллювиальные;
- морские;
- делювиальные;
- флювиогляциальные;
- эоловый;
- озерные.

Рисунок 5 – Виды песка

Основные виды песка и особенности их производства.

Сегодня практически все виды песка используются человеком в различных сферах деятельности и промышленности. Речной песок – это строительная смесь, которая добывается из русла реки. Этот материал обладает достаточно высокой степенью очистки, именно поэтому в его

структуре отсутствуют мелкие камни, примеси глинистого содержания и посторонние включения.

Карьерный песок добывается путем промывки водой в огромном количестве, в результате чего можно избавиться от пылевидных частиц глины. Этот материал достаточно распространен при изготовлении растворов, которые идут на закладку фундаментов и выполнение штукатурных работ, также его можно встретить и в асфальтовых смесях.

Строительный песок должен соответствовать ГОСТ 8736-2014, согласно которому материал представляет собой сыпучую неорганическую смесь из грубых зерен, размер которых достигает 5 мм. Плотность материала составляет 1300 кг/м^3. Строительный песок образуется при естественном разрушении горных пород, его добывают методами разработки песчано-гравийных и песчаных месторождений без применения и с применением оборудования для переработки полезных ископаемых.

К основным видам песка также относят искусственный песок, который имеет вид сыпучей смеси, получаемой путем механического дробления горных пород, среди последних следует выделить:

- шлаковый;
- гранитный;
- известняковый;
- мраморный;
- серпентинитовый;
- диоритовый;
- габбро;
- кварцевый;
- перлитовый.

Особенности искусственного песка.

Они могут иметь различное происхождение и плотность. Если сравнить зерна этого песка с зернами природного происхождения, то первые отличаются остроугольной формой и шероховатой поверхностью.

Искусственный песок обычно используют в качестве наполнителей при приготовлении штукатурок и декоративных растворов. В результате можно добиться ощутимой текстуры верхнего слоя на внешних поверхностях.

Этот материал может входить в состав любого слоя штукатурки, поскольку фракционная природа зерен может быть различной, в зависимости от вида раствора. Обычно размер зерна принимается равным размеру природного песка. При изготовлении искусственного песка в процессе переработки берется обожженный уголь, горная порода, а также несгоревшие частицы, в которых содержание серы низкое.

Характеристики материала будут зависеть от качества покрывающего слоя. При изготовлении декоративной штукатурки из такого песка для экономии может добавляться щебень, порошок из этой породы или крошка, от этого качество фактуры даже выигрывает.

Применение и характеристики морского песка.

Морской песок может быть использован при производстве строительных смесей, изготовлении заполнителей, выполнении штукатурных работ, закладке дорожных фундаментов, возведении ограждений и барьеров, в качестве наполнителя для строительных шпалер и красителей. Производство такого песка регламентируется ГОСТ 8736-93.

Фракции могут варьироваться от 2,5 до 3,5, что определяет модуль крупности частиц. Плотность зерна равна пределу от 2 до 2,8 г / $см^3$. В морском песке полностью примеси должны отсутствовать, но в некоторых фракциях можно обнаружить небольшое количество глины и пылевидных частиц. Морской песок характеризуется трудоемкостью добычи, что делает его стоимость выше, чем карьерный аллювиальный песок.

Применение и характеристики карьерного песка.

Главной особенностью карьерного песка является отсутствие примесей и периодичность. Аллювиальный карьерный материал имеет следующие характеристики: фракция от 1,5 до 5 мм, плотность равная 1,60 г / $см^3$, а также

низкое содержание глины, пыли и других примесей. Последний в своем составе должен составлять не более 0,03%.

Карьерный песок используется не только в строительстве, но и в отделке помещений, а также в народном хозяйстве. Особенно рентабельно использование такого песка в производстве бетона и кирпича, а также в дорожном и жилищном строительстве.

Карьерный песок может быть представлен в виде материала с фракцией от 2,5 до 2,7 мм. В производстве высокопрочных бетонных и железобетонных конструкций обычно используется крупнозернистый песок крупной фракции. Карьерный материал идет на кладку и изготовление тротуарной плитки.

Технические характеристики речных аллювиальных песков и особенности их добычи.

Данный вид имеет плотность 1,5 кг / м^3. Если же речь идет о плотности в состоянии естественной влажности, то эта цифра будет снижена до 1,45. Состав может содержать пылевидные частицы, иловые и глинистые элементы, но не более чем в объеме 0,7%. Содержание влаги в материале составляет 4%, в то время как удельный вес составляет 2,6 г / см^3. Эти виды песка добываются с помощью земснаряда, который закрепляется на барже. Такое оборудование дополняется гидромеханическими установками, мощными насосами, сетями и резервуарами для разделения материала по составу. Извлечение песка из русла высохших рек подобно извлечению карьерного песка.

Практически все виды песка можно отнести к первому классу радиоактивности. Исключение составляют только измельченные пески. Если говорить о других сортах, то они радиационно безопасны и могут быть использованы во всех строительных работах без ограничений.

Использование песка довольно распространено сегодня. Например, кварцевый песок используется для изготовления сварочных материалов общего и специального назначения. Пески также используются при отделочных работах, а также при ремонте помещений. Материал также

выступает в качестве компонента асфальтобетонных смесей, которые используются в дорожном строительстве.

1.2.3 Щепа

Древесная щепа – это ценный продукт, который получают в результате дробления или измельчения древесины, изображенный на рисунке 6.

Причем щепа может быть как отходами, появляющимися при обработке бревен или пиломатериала, так и конечным продуктом, который получают, измельчая древесину, не пошедшую в переработку или даже ее отходы.

Таким образом, применение щепы в различных сферах хозяйства позволяет решить проблему с утилизацией отходов деревообработки.

Отличие от других видов измельченной древесины.

Четкого отличия между щепой и другими видами мелких отходов, то есть опилками или стружкой, не существует, поэтому в большинстве случаев их различают по:

- способу получения;
- размеру;
- форме.

Рисунок 6 – Технологическое сырье – щепа

Основной способ получения этого вида отходов – станочная обработка (фрезерование) или ручная рубка/резка древесины, однако эти же методы нередко применяют и для получения стружки.

Четкого различия между стружкой и щепой не существует, поэтому принято считать, что при обработке деревянных деталей на фуговальных станках получается стружка, а после фрезерования щепа.

Также щепа получается после измельчения древесины на щепорезах, ведь по способу воздействия на древесные отходы они больше похожи на фрезерные станки.

В большинстве случаев форма измельченной древесины не оказывает существенного влияния на результат ее применения, однако исключением является материал, используемый для изготовления арбалита.

Ведь щепа выполняет в нем функции не только снижающего массу и поднимающего теплоизоляционные свойства наполнителя, но также арматуры, увеличивающей прочность готового изделия на изгиб.

Поэтому игольчатая щепа оказывается более эффективным наполнителем, чем любые опилки или стружка другой формы. При этом стружка игольчатой формы также эффективна, как и аналогичная щепа.

Основные характеристики.

К основным характеристикам этого продукта можно отнести:

- породу древесины;
- размер крупных, средних и мелких частиц;
- процентное содержание частиц различного размера;
- наличие коры;
- влажность;
- наличие поврежденной гнилью или больной древесины;
- плотность.

После измельчения щепа сохраняет все основные свойства исходной древесины, такие как:

- теплотворная способность;

- содержание различных веществ;
- способность впитывать воду;
- устойчивость к грибкам или болезням;
- прочность волокон.

Поэтому порода древесины, из которой получена щепа, нередко влияет на ее дальнейшее применение, делая этот материал пригодным или непригодным.

Размеры отдельных частиц, а также процентное соотношение между крупными, средними и мелкими, для некоторых применений является определяющим.

Еще один важный параметр – содержание коры, ведь теплотворная способность коры в десятки раз меньше, а химический состав сильно отличается от древесины.

Влажность также является важным параметром, от нее зависит вес в 1 $м^3$ щепы.

Кроме того, она влияет на теплотворную способность – чем выше влажность, тем меньше выделится тепла.

Если часть щепы сделана из поврежденной гнилью или больной древесины, то велика вероятность, что болезнетворные бактерии, находящиеся в них, заразят остальной материал.

Также поврежденный материал обладает заметно худшими теплоизоляционными и теплотворными способностями, а его волокна разрываются при меньшем усилии.

Сферы применения.

Существует несколько основных способов применения этого материала:

- копчение различных продуктов;
- настаивание алкогольных напитков;
- изготовление арболита и щепоцементных плит;
- сжигание в отопительных устройствах;
- декоративное/агротехническое мульчирование.

Изготовление щепы.

Основным материалом для производства этого продукта служат отходы лесодобывающих и лесоперерабатывающих предприятий. Любые обрезки и стволы, непригодные для делового использования, перерабатывают с помощью специальных устройств, которые называют щепорезами или дробилками для щепы.

Общий принцип работы этих устройств одинаков – острозаточенные ножи срезают небольшие кусочки измельчаемой детали, превращая их в щепу.

Благодаря тому, что ножи движутся быстро и по кругу, скорость, с которой щепорез измельчает древесину, довольно высока и может достигать значения 20–30 $м^3$ измельчаемого материала в час.

Все виды этих устройств можно разделить по типу измельчающего механизма на:

- дисковые (ножевые, роторные):
- барабанные;
- фрезерные.

Дисковые являются наиболее популярными благодаря простоте конструкции и высокой производительности, их основа – толстый стальной диск с прорезями, над которыми установлены острые ножи.

Барабанные несколько сложней и заметно тяжелей, но лучше подходят для переработки толстых стволов, их основа — пустотелый или цельный металлический барабан, из которого выступают острозаточенные ножи.

Фрезерные, которые также называют шредерами, применяют для грубого измельчения грязной или содержащей металлические фрагменты древесины.

Основа шредера – 1 или 2 толстых стальных вала, из которых выступают ножи различной формы.

Скорость вращения диска и барабана составляет 1 – 3 тысячи оборотов в минуту, благодаря чему они производят довольно мелкую щепу.

Шредер производит очень крупную, нередко с рваными краями, щепу, ведь скорость вращения его валов редко превышает 90 оборотов в минуту.

Из-за этого ножи не только режут, но также давят и дробят измельчаемую деталь, что и обеспечивает неровные формы с рваными краями у готового продукта [2].

1.3 Основные свойства, изучаемых материалов

1.3.1 Плотность

Плотность [20] – это масса единицы объема материала:

$$\rho = \frac{m}{V} \text{ (г/см}^3\text{, кг/м}^3\text{)},$$

где m – масса материала (г, кг),

V– объем материала (см^3, м^3).

Соотношение между размерностями плотности:

$$1 \text{ г/см}^3 = 1 \text{ т/м}^3 = 1000 \text{ кг/м}^3 = 1 \text{ кг/дм}^3 = 1 \text{ кг/л}.$$

Истинная плотность [20] – масса единицы объема кускового материала в абсолютно плотном состоянии (без пустот и пор). Для определения истинной плотности необходимо определить объем плотного материала (без пустот и пор). Для этого пористый материал предварительно измельчают до полной ликвидации пор. Истинная плотность численно равна отношению массы материала к его объему в абсолютно плотном состоянии (без пустот и пор):

$$\rho_И = \frac{m}{V_A} \text{ (г/см}^3\text{, кг/м}^3\text{)},$$

где m – масса материала (г, кг),

V_A – объем материала без пор (см^3, м^3).

Средняя плотность [20]– масса единицы объема кускового материала в естественном состоянии (с пустотами и порами), которая численно равна отношению массы образца материала к его объему в естественном состоянии (вместе с пустотами и порами):

$$\rho_C = \frac{m}{V_E} \text{ (г/см}^3\text{, кг/м}^3\text{)},$$

где m – масса материала (г, кг);

V_E – объем материала с пустотами и порами (см^3, м^3).

Большинство строительных материалов имеют поры. Чем их больше в материале, тем меньше его средняя плотность. При определении средней плотности материала можно использовать образцы как правильной, так и неправильной геометрической формы. От формы образца зависит метод определения его объема для последующего расчета средней плотности материала.

Насыпная плотность [20]– масса единицы объема сыпучего материала в естественно-насыпном состоянии:

$$\rho_{\text{Н}} = \frac{m}{V_{\text{Н}}} \text{ (г/см}^3\text{, кг/м}^3\text{)},$$

где m – масса материала (г, кг, т);

V_H– объем материала в насыпном состоянии (см^3, м^3).

1.3.2 Пористость

Пористость материала [20] характеризует степень заполнения его объема воздушными порами и численно равна отношению объема пор $V_П$ к объему образца в естественном состоянии V_E:

$$П = \frac{V_П}{V_E}.$$

Пористость $П$ кускового материала связана с его средней плотностью ρ_C и истинной плотностью $\rho_И$ следующим соотношением:

$$П = \frac{V_П}{V_E} = \frac{E_E - V_A}{V_E} = \frac{V_E - V_A}{V_E} = 1 - \frac{V_A}{V_E} = 1 - \frac{V_A \cdot m_1}{V_E \cdot m_1};$$

$$П = 1 - \frac{\left(\frac{m_1}{V_E}\right)}{\left(\frac{m_1}{V_A}\right)} = 1 - \frac{\rho_C}{\rho_И};$$

$$П = \left(1 - \frac{\rho_C}{\rho_И}\right) 100\%;$$

где $V_П$ – объем пор в куске материала, см^3;

V_E – объем материала в естественном состоянии (с порами), см^3;

V_A– объем материала в абсолютно плотном состоянии, см^3;

m_1 – масса образца материала в сухом состоянии, г;

ρ_C– средняя плотность кускового материала, г/см^3;

$\rho_И$– истинная плотность кускового материала, г/см^3.

Пористость выражают в долях или в процентах от объема материала. Пористость и ее вид (открытая и закрытая) в значительной степени определяет свойства материалов: водопоглощение, водопроницаемость, морозостойкость, теплопроводность, плотность.

Открытая пористость $П_О$ [20] равна отношению суммарного объема всех открытых пор V_O, насыщающихся водой, к объему материала в естественном состоянии V_E:

$$П_0 = \frac{V_0}{V_E} = \frac{V_В}{V_E} = \frac{\left(\frac{m_в}{\rho_в}\right)}{V_e} = \frac{\left(\frac{m_2 - m_1}{\rho_в}\right)}{V_e};$$

$$П_0 = \left(\frac{m_2 - m_1}{\rho_в \cdot V_E}\right) 100\%;$$

где $П_О$ – открытая пористость кускового материала в долях или процентах;

V_O – объем открытых пор в куске материала, см^3;

V_E – объем материала в естественном состоянии (с порами), см^3;

m_1 – масса образца материала в сухом состоянии, г;

m_2 – масса образца материала, насыщенного водой, г;

m_B – масса воды, поглощенной образцом материала, г;

ρ_B – истинная плотность воды, г/см^3.

Открытую пористость выражают в долях или в процентах от объема материала.

Закрытая пористость $П_З$ равна разности между общей пористостью $П$ и открытой пористостью $П_О$:

$$П_З = П - П_0;$$

$$П_З = П - П_0 = \left(1 - \frac{\rho_C}{\rho_И}\right) - \left(\frac{m_2 - m_1}{\rho_в \cdot V_E}\right);$$

$$П_З = П - П_0 = \left(1 - \frac{\rho_C}{\rho_И}\right) 100\% - \left(\frac{m_2 - m_1}{\rho_в \cdot V_E}\right) 100\%.$$

Коэффициент плотности $К_{ПЛ}$ [20] характеризует степень заполнения объема материала твердым веществом и равен отношению объема материала в абсолютно плотном состоянии V_A к объему материала в естественном состоянии V_C:

$$К_{ПЛ} = \frac{V_А}{V_E} = \frac{V_А \cdot m_1}{V_Е \cdot m_1} = \frac{\left(\frac{m_1}{V_E}\right)}{\left(\frac{m_1}{V_E}\right)} = \frac{\rho_C}{\rho_И};$$

$$К_{ПЛ} = \frac{\rho_C}{\rho_И} \leq 1.$$

Коэффициент насыщения пор $К_Н$ равен отношению открытой пористости $П_О$ к общей пористости $П$ и характеризует степень насыщения объема материала водой:

$$К_Н = \frac{П_О}{П} \leq 1.$$

Коэффициент насыщения пор может изменяться от 0 (стекло, металл, гранит – все поры в материале замкнутые) до 1 (минеральная или стеклянная вата – все поры открытые).

1.3.3 Гигроскопичность

Гигроскопичность $W_Г$ [21] – свойство материала поглощать водяной пар из воздуха. Этот физико-химический процесс называется сорбцией. Он является обратимым и повышается с повышением давления водяного пара (с увеличением относительной влажности воздуха при постоянной температуре).

$$W_Г = \frac{m_2 - m_1}{m_1} 100\%,$$

где $W_Г$ – гигроскопическая влажность, доли или проценты;

m_1 –масса образца в сухом состоянии, г;

m_2 –масса образца в увлажненном состоянии, г.

Это обусловлено осаждением водяного пара на внутренних поверхностях пор и капиллярной конденсацией.

1.3.4 Водопоглощение

Водопоглощение [21] – способность материала поглощать и удерживать в порах воду. По величине водопоглощения можно характеризовать открытую пористость материала. При этом часть открытых мелких пор и все замкнутые поры водой не насыщаются.

Водопоглощение подразделяют на два вида – по массе W_m и по объему W_V.

Водопоглощение по массе W_m равно отношению массы воды m_B, поглощенной образцом при насыщении его водой, к массе сухого образца m_1:

$$W_m = \frac{m_{\text{В}}}{m_1} = \frac{m_2 - m_1}{m_1};$$

$$W_m = \frac{m_2 - m_1}{m_1} 100\%,$$

где W_m– водопоглощение по массе, доли или проценты;

m_B –масса воды, поглощенной образцом, г;

m_1 –масса образца в сухом состоянии, г;

m_2 –масса образца в насыщенном водой состоянии, г.

Водопоглощение по массе W_m выражают в долях или в процентах.

Водопоглощение по объему W_V [21] равно отношению объема поглощенной образцом воды V_B к объему образца в естественном состоянии V_E (вместе с порами и пустотами):

$$W_V = \frac{V_{\text{В}}}{V_E} = \frac{\left(\frac{m_{\text{В}}}{\rho_{\text{В}}}\right)}{V_E} = \frac{\left(\frac{m_2 - m_1}{\rho_{\text{В}}}\right)}{V_E} = \frac{m_2 - m_1}{V_E \cdot \rho_{\text{В}}};$$

$$W_V = \frac{m_2 - m_1}{V_E \cdot \rho_{\text{В}}} 100\%,$$

где W_V – водопоглощение по объему, доли или проценты;

m_1 – масса образца материала в сухом состоянии, г;

m_2 – масса образца материала, насыщенного водой, г;

m_B – масса воды, поглощенной образцом материала, г;

V_E – объем материала в естественном состоянии, см^3;

ρ_B – истинная плотность воды, равная 1 г/см^3.

$$\frac{W_V}{W_m} = \frac{\left(\frac{m_2 - m_1}{V_E \cdot \rho_{\text{В}}}\right)}{\left(\frac{m_2 - m_1}{m_1}\right)} = \frac{m_1}{V_E \cdot \rho_{\text{В}}} = \frac{\left(\frac{m_1}{V_E}\right)}{\rho_{\text{В}}} = \frac{\rho_{\text{С}}}{\rho_{\text{В}}};$$

$$\frac{W_V}{W_m} = \frac{\rho_{\text{С}}}{\rho_{\text{В}}}.$$

Водопоглощение по объему W_V выражают в долях или в процентах. Отношение водопоглощения по объему W_V к водопоглощению по массе W_m равно отношению средней плотности материала ρ_C к плотности воды ρ_B.

1.3.5 Коэффициент размягчения

Коэффициент размягчения K_P [21] – отношение прочности материала, насыщенного водой R_B, к прочности сухого материала R_C:

$$\text{К}_\text{Р} = \frac{R_{\text{В}}}{R_{\text{С}}} \leq 1.$$

Коэффициент размягчения характеризует водостойкость строительных материалов. При коэффициенте размягчения 0,8 и выше материал считается водостойким, менее 0,8 – не водостойким.

1.3.6 Коэффициент конструктивного качества

Коэффициент конструктивного качества $К_{КК}$ [21] равен отношению прочности материала R к его приведенной плотности d:

$$К_{КК} = \frac{R}{d};$$

где $К_{КК}$ – коэффициент конструктивного качества;

R – прочность материала, МПа;

d – приведенная плотность материала.

Приведенная плотность d – это отношение средней плотности материала ρ_C к плотности воды ρ_B:

$$d = \frac{\rho_С}{\rho_В};$$

где d – приведенная плотность;

ρ_C – средняя плотность материала, г/см3;

ρ_B – плотность воды, равная 1 г/см3.

Коэффициент конструктивного качества характеризует эффективность использования данного вида материала из серии подобных [3].

1.4 Технические характеристики сырья, влияющие на истинный объем

1.4.1 Щебень. Вид, фракционность, лещадность, водопоглащение

Для определения объема нужно перемножить насыпную плотность нерудного материала, длину и ширину участка, который предстоит засыпать, а также толщину щебневого слоя.

Насыпной плотностью сыпучего материала называют его плотность в естественном неуплотненном (рыхлом) состоянии. Параметр насыпной плотности щебня всегда меньше, чем показатель обычной плотности, так как

на него влияют размеры зерен щебня и степень пустотности между ними. Обычно принимают усредненный показатель насыпной плотности, равный 1,4 т/куб. м.

Величина насыпной плотности щебня зависит от следующих факторов:

1. Вида горной породы, используемой при производстве щебня (Например, известняк - самый легкий из-за слоистой и пористой структуры, а гранит - наиболее тяжелый ввиду высокой плотности).

2. Размера фракции щебня (рисунок 7).

Это условное разделение щебня на виды по размеру его зерна, представленные в таблице 1. Существуют – основные фракции щебня, которые упоминаются в соответствующих нормативных документах (ГОСТ).

Таблица 1 – Насыпная плотность в зависимости вида и фракционности щебня

Щебень	Размер фракции, мм	Насыпная плотность, кг/м^3
Гранитный	20 – 40	1370 – 1470
	40 – 70	1380 – 1450
	70 – 250	1400
Известняковый	10 – 20	1250
	20 – 40	1280
	40 – 70	1330
Гравийный	0 – 5	1600
	5 – 20	1430
	20 – 40	1400
	40 – 100	1650
	больше 160	1730
Шлаковый	независимо от размера частиц	800

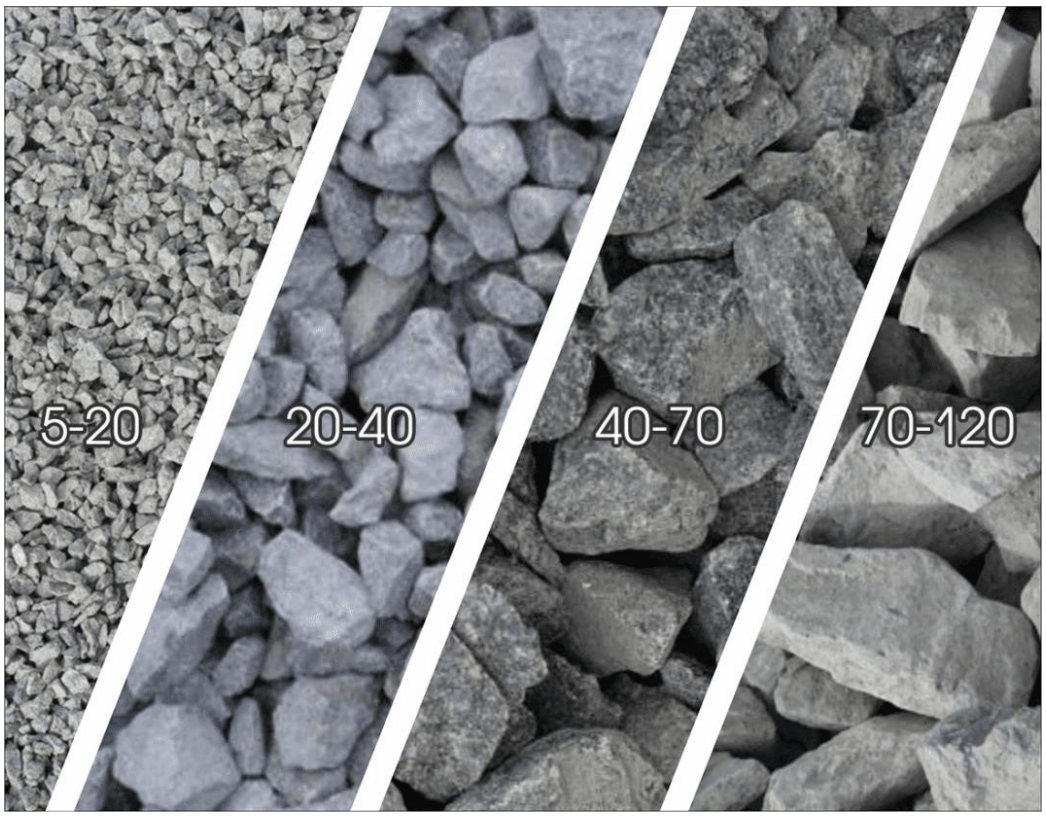

Рисунок 7 – Некоторые виды фракций щебня

3. Формы зерна щебня, а именно лещадность.

Лещадность щебня представляет собой величину, которая характеризует процентное содержание плоских и угловатых частиц в общем объеме щебня, разновидность которых видно на рисунке 8 и представленные в таблице 2. Зерна, которые имеют длину в три раза больше толщины, будут считаться пластинчатыми. А игловатыми принято считать зерна, у которых длина в три раза больше ширины.

Таблица 2 – Лещадность щебня (5 групп)

Группа		Содержание зерен пластинчатой и игловатой форм
I	кубовидная	до 10 %
II	улучшенная	от 10 % до 15 %
III	обычная	от 15 % до 25 %
IV	обычная	от 25 % до 35 %
V	обычная	от 35 % до 50 %

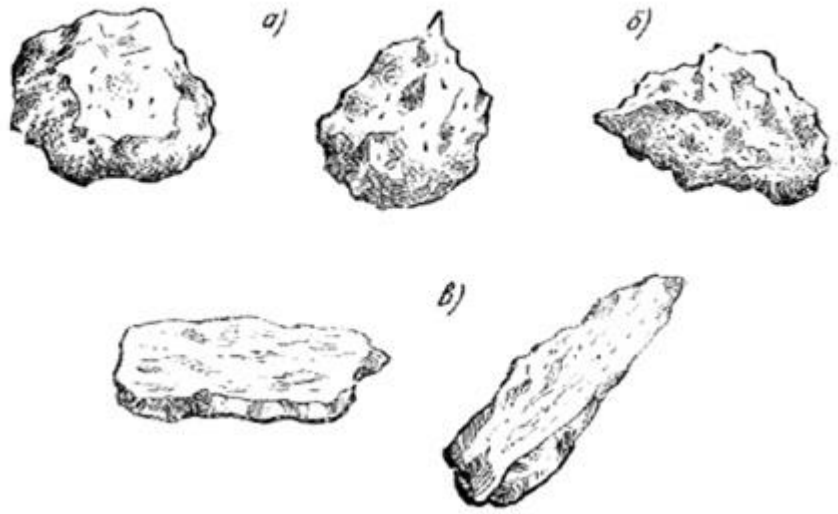

Рисунок 8 - Схема разновидностей щебня: А – кубовидный, Б – остроугольный, В – пластинчатый/игловатый

4. Водопоглощения щебня.

Водопоглощением любого строительного материала называют его способность при длительном погружении в жидкость впитывать влагу и удерживать ее в порах. Численно показатель водопоглощения выражается в процентах относительно массы или объема сухого образца, характеризуя максимальное количество жидкости, которое способен поглотить материал до полного насыщения. Высокая степень водопоглощения ухудшает физико-механические свойства материала, увеличивая его среднюю плотность и теплопроводность и снижая прочность.

Водопоглощение горных пород, идущих в производство щебня, меняется в широком диапазоне – от (0,02-0,7) % для гранита до 34% для туфа. Показатели водопоглощения природного щебня полностью идентичны соответствующим характеристикам исходных горных пород, то есть водопоглощение гранитного щебня много ниже, чем у щебня известнякового. В общем случае водопоглощение щебня зависит от плотности горной исходной породы, от конфигурации поровых каналов, пронизывающих щебневые зерна, от размеров фракции самих зерен. Для щебня из плотных горных пород степень водопоглощения определяется удельной поверхностью

зерна – чем меньше размер фракции, тем больше возрастает удельная поверхность зерен и увеличивается адсорбционная способность. Количество влаги, образующей водную пленку на поверхности зерен, превышает объемы влаги, заполнившей поры [14].

1.4.2 Песок. Фракционность, плотность, уровень влажности, класс

На вычисление объема песка влияют:

1. Размер фракций (рисунок 9).

Фракция — средняя величина частиц. Чтобы ее определить, сыпучий материал просеивают через специальные сита. По этому признаку в таблице 3 можно увидеть, как песок подразделяется.

Таблица 3 – Классификация песков по зерновому составу

Группа песка	Модуль крупности	Полный остаток на сите №063, % по массе
Крупный	Свыше 2,5	Свыше 45
Средний	2,0…2,5	30…45
Мелкий	1,5…2,0	10…30
Очень мелкий	1,0…1,5	До 10

Рисунок 9 – Фракции песка

2. Плотность.

Это соотношение массы песка и его объема. Иными словами, характеристика показывает, сколько килограмм в кубометре материала. Здесь важно сказать, что конкретный показатель зависит от нескольких факторов.

На насыпную плотность влияют:

- пористость;
- влажность.

Так, если везут не утрамбованный сухой песок, то его насыпная плотность будет значительно ниже, чем если бы привезли утрамбованный материал, да еще и не высохший после дождя.

Таким образом, заранее узнать насыпную плотность невозможно. Для каждой отдельной партии она будет отличаться. Обычно используется среднее значение, с учетом основных характеристик материала.

Например, для песка средняя насыпная плотность будет следующей:

- при влажности до 2% – 1 150 кг/м^3;
- при влажности до 5% – 1 180 кг/м^3;
- при влажности до 10% – 1 220 кг/м^3;
- при влажности до 15% – 1 500 кг/м^3;
- при влажности до 20% – 1 890 кг/м^3;
- при влажности до 30% – 2 160 кг/м^3.

Но чаще всего берут совсем усредненные показатели, колеблющиеся в пределах 1 300-1 500 кг/м^3.

3. Уровень влажности.

Название говорит само за себя. Это процентное количество влаги, содержащееся в песке. Разумеется, это не статичный показатель. Влажность может меняться в зависимости от степени просушки песка, условий его хранения, климатической обстановки и прочих факторов.

При этом, для некоторых областей применения песка существуют четкие требования к влажности поставляемой продукции.

Например:

- для приготовления сухих цементных смесей допускается влажность до 5%;
- для приготовления бетона влажность тоже не должна превышать 5%. В противном случае приходится менять пропорции добавления воды в раствор. Кстати, строители умеют определять влажность на глаз. Для этого берется горсть песка и сжимается в кулаке. Если после этого она остается в виде комка и не рассыпается, то влажность более 5%.
- А вот для песочниц, используемых в железнодорожных составах для сокращения тормозного пути, предел влажности песка – всего 0,5%. Если этот показатель будет выше, то зерна не смогут создать достаточного сцепления.

4. Класс песка.

Этот параметр относится к качеству зернового состава материала.

Всего выделяют 2 класса:

- I класс – более качественный;

- II класс – менее качественный.

Песок I класса.

Он обладает более однородным составом и меньшим процентом содержания вредных примесей.

Например, в нем должно присутствовать не более 0,5% зерен крупностью более 10 мм.

Допустимое содержание пылевидных и глинистых частиц у такого песка – не более 2% для крупных фракций и не более 3% – для мелких.

Глины в комках должно быть не более 0,25% для крупных фракций и не более 0,35% – для мелких.

Песок II класса.

Здесь допускается менее однородный состав и большее содержание вредных примесей.

Для сравнения, у данного песка может быть до 5% зерен размером более 10 мм (для крупных фракций) и до 0,5% (для мелких фракций).

Пылевидных и глинистых частиц может содержаться до 3% (для крупных фракций) и до 10% (для мелких фракций).

Глины в комках может быть до 0,5% (для крупных фракций) и до 1% (для мелких фракций).

Согласно требованиям ГОСТа, предъявляемые к материалам для строительных работ, песок I класса идет на более ответственные работы (фундамент, несущие конструкции, инженерные сооружения). Для менее серьезных работ подойдет продукция II класса [15].

1.4.3 Щепа. Плотность, породный состав, влажность

В промышленности и в отопительных целях используют измельчённую древесину – древесную щепу, опилки и стружку. Насыпная плотность измельчённой древесины определяется степенью её измельчения, влажностью древесной смеси и породой измельчённых деревьев, что можно увидеть в

таблице 4. Определяющим фактором плотности измельчённой древесной массы выступает фракционный состав – степень измельчения древесного материала, пример показан рисунке 10.

Таблица 4 – Плотность щепы и опилок в зависимости от породы дерева

Порода дерева	Насыпная плотность свежеотгруженной технологической щепы		Насыпная плотность свежеотгруженных древесных опилок	
	Плотность ($кг/м^3$)	Предел плотности ($кг/м^3$)	Плотность ($кг/м^3$)	Предел плотности ($кг/м^3$)
Дуб	292	248-371	227	193-288
Акация	277	234-288	215	182-225
Граб	273	266-286	213	207-223
Ясень	270	187-342	210	146-266
Рябина (дерево)	262	248-320	204	193-249
Яблоня	259	237-302	202	185-235
Бук	244	223-295	190	174-230
Вяз	238	202-295	185	157-230
Лиственница	239	194-239	186	151-186
Клён	236	205-248	183	160-193
Берёза	234	184-277	182	143-216
Груша	241	211-256	188	164-199
Каштан	234	216-259	182	168-202
Кедр	205	202-209	160	157-162
Сосна	187	112-274	146	87-213
Липа	184	158-288	143	123-224
Ольха	180	169-209	140	132-162
Ива	176	167-212	137	129-165
Осина	169	166-198	132	129-154
Ель	162	133-270	126	104-210
Верба	162	151-180	126	118-140
Орех лесной	155	151-162	120	118-126
Орех грецкий	202	176-212	157	137-165
Тополь	153	140-212	119	109-165
Пихта	148	126-216	115	98-168

Породный состав – количественное соотношение древесины разных пород, исчисляется в процентном содержании породы во всей массе древесной смеси. При расчёте плотности измельчённой смешанной древесной массы, коэффициенты породности применяются совместно со значениями величины плотности для древесины соответствующей породы.

Размер частицы – длине соответствует размер вдоль волокон, ширине – наибольший размер поперёк волокон, толщине – наименьший размер поперёк волокон (таблица 5).

Рисунок 10 – Разновидность форм и фракций щепы

Фракция – совокупность древесных частиц, близким по своим геометрическим размерам. Максимальный размер для частицы фракции определяется по наибольшей её стороне.

Таблица 5. Требования к размерам щепы в зависимости от направления использования

Назначение щепы	Размеры щепы, мм		
	длина	оптимальная длина	толщина не более
Для ЦБП	15-25	18	
Для ДВП	10-35	25	5
Для ДСП			530
Плоского прессования	20-60	40	30
Экструзионного прессования	5-40	20	5
Для гидролизного производства	5-35	20	

Влажность так же является важной характеристикой щепы.

Массовая доля воды в массе измельчённой древесины. Состоит из влаги, содержащейся в частичках самой древесины и влаги, находящейся между ними – поглощённой смесью, впитанной (абсорбированной) воды. За счёт впитанной влаги, в отличие от влажности древесины – влажность измельчённой древесной массы может достигать огромных цифр, например 150...200%. В древесной массе, влажность от 30% начинаются грибковые гнилостные процессы. Поэтому, измельчённая древесина, влажностью до 30% принимается, как пригодная к хранению. Остальную же древесину нужно сушить [16].

1.5 Математическое обеспечение действующих измерительных комплексов и их функциональность

Электронный тахеометр при наведении на снимаемую точку измеряет три величины:

- наклонное расстояние;
- горизонтальное направление;
- вертикальный угол.

Чтобы понять смысл измеряемых величин, рассмотрим на примере одной станции схему измерений, представленную на рисунке 11.

На схеме изображены: точка стояния тахеометра S и две снимаемых точки P_1 и P_2. Прибор выведен в рабочее положение. Это означает, что вертикальная ось вращения прибора занимает отвесное положение (направлена в зенит), а плоскость горизонтального круга находится в горизонтальном положении. Нулевой отсчет лимба горизонтального круга занимает произвольное положение либо можно считать, что он сориентирован по какому-то начальному направлению.

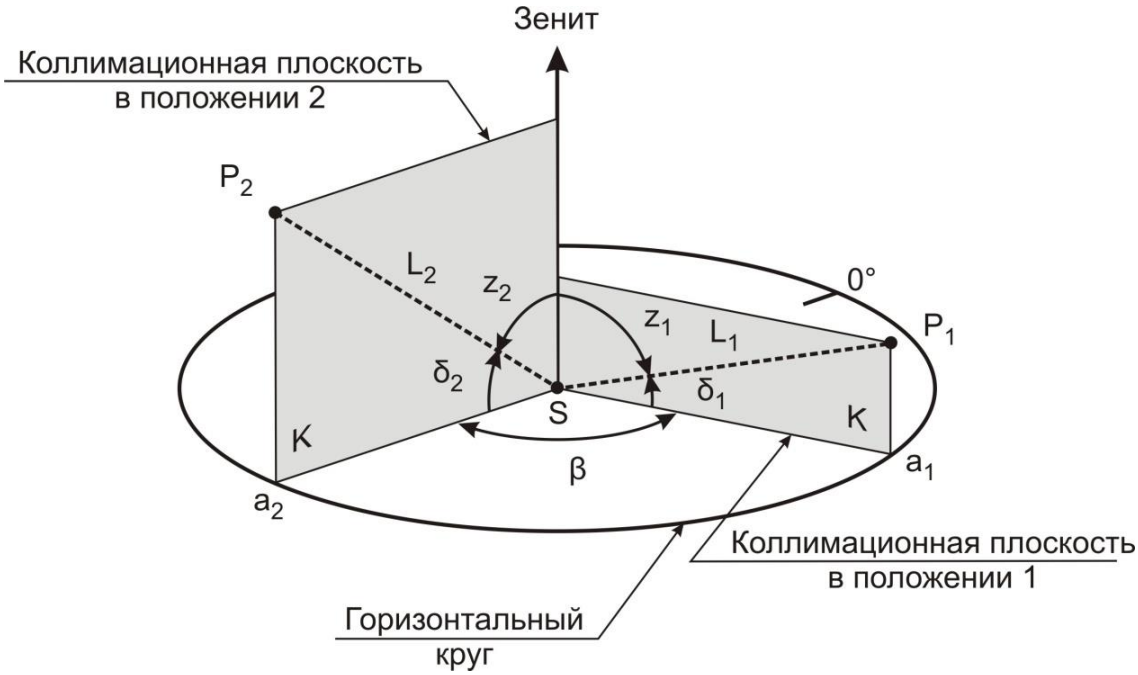

Рисунок 11– Схема измерений

Для выполнения измерений на точку P_1 визирная ось зрительной трубы должна быть наведена на эту точку. С геометрической точки зрения это означает, что коллимационная плоскость K прибора должна быть совмещена с точкой P_1.

В этом случае все три измеряемых величины будут лежать в одной плоскости. Только после этого можно выполнять измерения на точку P_1.

Аналогичным образом для выполнения измерений на точку P_2 коллимационная плоскость K прибора должна быть совмещена с точкой P_2

Длина линии.

Наиболее простой смысл имеет длина линии L. Это измеренное наклонное расстояние L_0 между точкой излучения сигнала в тахеометре и точкой отражения сигнала на отражателе, исправленное поправкой за атмосферные условия ΔL_{ATM} и постоянным слагаемым дальномера C:

$$L = L_0 + \Delta L_{ATM} + C.$$

Поправка за атмосферные условия ΔL_{ATM} вычисляется как

$$\Delta L_{ATM} = ppm_{ATM} \cdot 10^{-6} \cdot L_{MM}.$$

где ppm_{ATM} - поправка в мм на 1 км длины линии;

L_{MM} – длина измеряемой линии, выраженная в миллиметрах.

В тахеометрах в качестве рабочей формулы для вычисления исправленного значения длины линии L используется формула

$$L = L_0 \cdot (1 + ppm_{ATM} \cdot 10^{-6}) + C,$$

получающаяся в результате подстановки. Все величины выражены в миллиметрах.

На дисплей прибора выводится исправленное значение длина линии L.

Угловые величины [22].

Наибольшее недопонимание у студентов вызывают угловые величины.

При измерении угловых величин тахеометром важно понимать и помнить, что в горизонтальной плоскости измеряются горизонтальные направления, а в вертикальной плоскости – вертикальные углы.

Угол (имеется в виду плоский угол) – это часть плоскости, заключенной между двумя лучами, лежащими в этой плоскости и выходящими из одной точки (вершины). То есть для образования угла всегда необходимо два луча или два направления. Если этим двум направлениям соответствуют какие-то два отсчета O_1 и O_2, взятые по неподвижному измерительному кругу, находящемуся в одной плоскости с указанными лучами и центр которого совмещен с вершиной лучей, образующих угол, то тогда величина u этого угла может быть определена как

$$u = O_2 - O_1.$$

Здесь можно провести аналогию с отрезками прямых. Для того, чтобы говорить об отрезках прямых необходимо первоначально ограничить эти

прямые двумя параллельными друг другу линиями. Только в этом случае получится отрезок. Никакая одна прямая, пересекающая другую прямую, не может в отдельности образовать отрезок. Собственно, отрезок – это дуга, на которую опирается центральный угол, вершина которого находится в бесконечности.

Горизонтальное направление [22].

Горизонтальное направление *a* – это отсчет по горизонтальному кругу тахеометра, соответствующий тому положению коллимационной плоскости *K* прибора, при котором она совмещена с измеряемой точкой. Это не угол, это просто отсчет.

В случае измерения угловых величин в горизонтальной плоскости, при наведении визирной оси на точку можно говорить только об измерении горизонтального направления на точку, взятии отсчета по горизонтальному кругу, но не об измерении угла, так как нет второго горизонтального направления, нет второго отсчета. Чтобы получить второй отсчет надо развернуть коллимационную плоскость на вторую точку. Но это будет уже второе наведение, второе положение коллимационной плоскости. А одному положению коллимационной плоскости всегда соответствует только один отсчет по горизонтальному кругу.

На рисунке, представленном выше, направление на точку P_1, соответствующее ему положение коллимационной плоскости K_1 и отсчет по горизонтальному кругу a_1. Аналогичные элементы и величины показаны и для точки P_2.

Основные отличия лазерного сканера от любых традиционных тахеометров [22] – гораздо большая скорость измерений, полностью автоматизированный сервопривод, поворачивающий измерительную головку в обеих (как горизонтальной, так и вертикальной) плоскостях и, самое главное, – скорость (до 5000 измерений в секунду, или в среднем – два-три полных рабочих дня измерений обычным тахеометром) и плотность (до десятков точек на 1 квадратный сантиметр поверхности). Полученная после измерений 3*D*

модель объекта представляет собой гигантский набор точек (от сотен тысяч до нескольких миллионов), имеющих координаты с высочайшей, миллиметровой точностью.

Принцип работы 3*D* лазерного сканера тот же, что и обыкновенного тахеометра – измерение расстояния до объекта и двух углов, что, в конечном итоге, дает возможность вычислить координаты. Лазерный пучок исходит из излучателя, отражается от поверхности объекта и возвращается в приемник. Вращающаяся призма (или зеркало) распределяет пучок по вертикали с заранее заданным шагом (например, в 0,1°). Таким образом, в отдельно взятом вертикальном скане будут измерены все точки с дискретностью в 0,1° (так, при максимальном вертикальном угле 3*D* сканирования в 140° их будет, соответственно, 1400). Затем сервопривод автоматически поворачивает блок измерительной головки на угол, равный шагу измерения (при той же дискретности в 0,1° полный оборот сканера состоит из 3600 отдельных вертикальных плоскостей). В итоге полная цифровая картина окружающего пространства будет представлена в виде набора из 5040000 точек. Пять миллионов точек с высокой точностью за 30 минут работы. Как правило, весь процесс съемки полностью автоматизирован. Данные измерений в реальном времени записываются на внешний или внутренний носитель.

После того, как произведены все измерения, начинается процесс обработки полученных данных. Изначально «сырые измерения» представляют собой набор («облако») точек, который необходимо представить в виде чертежей или схем. Процесс обработки зависит от желаемого конечного результата, от того, что конкретно нужно получить: это может быть непосредственно и само облако точек, триангуляционная поверхность (ТИМ), набор сечений, план, сложная трехмерная модель, либо просто набор измерений (длины, периметры, диаметры, площади, объемы). [8]

1.6 Возможности расширения функциональности с применением детерминированных и стохастических моделей и методов, платформы asp.net.core и web разработки

1.6.1 Платформа ASP.NET Core

Платформа *ASP.NET Core* [18] представляет технологию от компании *Microsoft*, предназначенную для создания различного рода веб-приложений: от небольших веб-сайтов до крупных веб-порталов и веб-сервисов.

С одной стороны, *ASP.NET Core* является продолжением развития платформы *ASP.NET*. Но, с другой стороны, это не просто очередной релиз. Выход *ASP.NET Core* фактически означает революцию всей платформы, ее качественное изменение.

Разработка над платформой началась еще в 2014 году. Тогда платформа условно называлась *ASP.NET vNext*. В июне 2016 года вышел первый релиз платформы. А в декабре 2019 года вышла версия *ASP.NET Core* 3.1.

ASP.NET Core теперь полностью является *opensource*-фреймворком. Все исходные файлы фреймворка доступны на *GitHub*.

ASP.NET Core может работать поверх кроссплатформенной среды *.NET Core*, которая может быть развернута на основных популярных операционных системах: *Windows, Mac OS, Linux*. И таким образом, с помощью *ASP.NET* Core мы можем создавать кроссплатформенные приложения. И хотя *Windows* в качестве среды для разработки и развертывания приложения до сих пор превалирует, но теперь уже мы не ограничены только этой операционной системой. То есть мы можем запускать веб-приложения не только на ОС *Windows*, но и на *Linux* и *Mac OS*. А для развертывания веб-приложения можно использовать традиционный *IIS*, либо кроссплатформенный веб-сервер *Kestrel*.

Благодаря модульности фреймворка [18] все необходимые компоненты веб-приложения могут загружаться как отдельные модули через пакетный

менеджер *Nuget*. Кроме того, в отличие от предыдущих версий платформы нет необходимости использовать библиотеку *System.Web.dll*.

ASP.NET Core включает в себя фреймворк *MVC*, который объединяет функциональность *MVC, Web API* и *Web Pages*. В предыдущих версии платформы данные технологии реализовались отдельно и поэтому содержали много дублирующей функциональности. Сейчас же они объединены в одну программную модель *ASP.NET Core MVC*. А *Web Forms* полностью ушли в прошлое.

Кроме объединения вышеупомянутых технологий в одну модель в MVC был добавлен ряд дополнительных функций.

Одной из таких функций являются тэг-хелперы (*tag helper*), которые позволяют более органично соединять синтаксис *html* с кодом *C#*.

ASP.NET Core характеризуется расширяемостью. Фреймворк построен из набора относительно независимых компонентов. И мы можем либо использовать встроенную реализацию этих компонентов, либо расширить их с помощью механизма наследования, либо вовсе создать и применять свои компоненты со своим функционалом.

Также было упрощено управление зависимостями и конфигурирование проекта. Фреймворк теперь имеет свой легковесный контейнер для внедрения зависимостей, и больше нет необходимости применять сторонние контейнеры, такие как *Autofac, Ninject*. Хотя при желании их также можно продолжать использовать.

В качестве инструментария разработки мы можем использовать последние выпуски *Visual Studio*, начиная с версии *Visual Studio* 2015. Кроме того, мы можем создавать приложения в среде *Visual Studio Code*, которая является кроссплатформенной и может работать как на *Windows*, так и на *Mac OS X и Linux*.

Для обработки запросов теперь используется новый конвейер *HTTP*, который основан на компонентах *Katana* и спецификации *OWIN*. А его модульность позволяет легко добавить свои собственные компоненты.

Если суммировать [18], то можно выделить следующие ключевые отличия *ASP.NET Core* от предыдущих версий *ASP.NET*:

- новый легковесный и модульный конвейер *HTTP*-запросов;
- возможность развертывать приложение как на *IIS*, так и в рамках своего собственного процесса;
- использование платформы *.NET Core* и ее функциональности;
- распространение пакетов платформы через *NuGet;*
- интегрированная поддержка для создания и использования пакетов *NuGet;*
- единый стек веб-разработки, сочетающий *Web UI* и *Web API;*
- конфигурация для упрощенного использования в облаке;
- встроенная поддержка для внедрения зависимостей;
- расширяемость;
- кроссплатформенность: возможность разработки и развертывания приложений *ASP.NET* на *Windows*, *Mac* и *Linux;*
- развитие как *open source*, открытость к изменениям.

Эти и другие особенности и возможности стали основой для новой модели программирования.

Структура проекта *ASP.NET Core*.

Рассмотрим базовую структуру стандартного проекта *ASP.NET Core*. Проект *Empty* содержит очень простую структуру - необходимый минимум для запуска приложения:

- *connected Services*: подключенные сервисы из *Azure;*
- *dependencies*: все добавленные в проект пакеты и библиотеки, иначе говоря зависимости;
- *properties*: узел, который содержит некоторые настройки проекта. В частности, в файле *launchSettings.json* описаны настройки запуска проекта, например, адреса, по которым будет запускаться приложение;
- *appsettings.json*: файл конфигурации проекта в формате *json;*

- *program.cs*: главный файл приложения, с которого и начинается его выполнение. Код этого файла настраивает и запускает веб-хост, в рамках которого разворачивается приложение;

- *startup.cs*: файл, который определяет класс *Startup* и который содержит логику обработки входящих запросов.

1.6.2 Краткая конструкция приложения

Приложение *ASP.NET Core* – это просто консольное приложение, которое создает веб сервер в своем методе *Main*:

Main использует *:dn: cls:`~Microsoft. AspNetCore. Hosting. WebHostBuilder`*, который следует паттерну сборки для создания хоста веб приложения. У паттерна есть методы, которые определяют веб сервер (например, *UseKestrel*) и класс для запуска (*UseStartup*). В примере выше используется веб сервер *Kestrel*, но мы можем указать и другие серверы. В следующем разделе мы подробнее рассмотрим UseStartup. WebHostBuilder предлагает много дополнительных методов, включая *UseIISIntegration* для хостинга на *IIS* и *IIS Express* и *UseContentRoot* для указания корневой директории контента. Методы *Build* и *Run* создают *IWebHost*, который будет хостить приложение, и оно начнет слушать входящие *HTTP* запросы.

Startup

Метод *UseStartup* для *WebHostBuilder* указывает класс *Startup* для вашего приложения.

В классе *Startup* вы определяете поток обработки запросов, а также здесь настраиваются все сервисы, необходимые вашему приложению. Класс *Startup* должен быть открытым и содержать следующие методы:

- *configureServices* определяет, используемые вашим приложением (например, *ASP.NET MVC Core, Entity Framework Core, Identity* и так далее)

- *configure* определяет связующее ПО в потоке запросов

Сервисы.

Сервис – это компонент для общего пользования в приложении. Сервисы доступны благодаря внедрению зависимостей. *ASP.NET Core* включает в себя встроенный *IoC* контейнер, который по умолчанию поддерживает внедрение конструктора, но вы можете легко заменить его *IoC* контейнером по вашему выбору. В дополнение к преимуществу слабого связывания, *DI* делает так, что сервисы доступны всему приложению.

Связующее ПО (программное обеспечение).

В *ASP.NET Core* составляется поток запросов, используя связующее ПО (*Middleware*). Связующее ПО *ASP.NET Core* выполняет асинхронную логику для *HttpContext*, а затем либо вызывает следующее связующее ПО в цепочки, либо напрямую обрывает запрос. Обычно для связующего ПО используется «*Use*», принимая зависимость для пакета *NuGet* и вызывая соответствующий метод расширения *UseXYZ* для *IApplicationBuilder* в методе *Configure*.

ASP.NET Core предлагает богатый набор связующего ПО:

- статические файлы;
- роутинг;
- аутентификация.

Также вы можете использовать пользовательское связующее ПО.

С *ASP.NET Core* можно использовать любое связующее ПО, основанное на *OWIN*.

Серверы.

Хостинговая модель *ASP.NET Core* [18] напрямую не слушает запросы – она полагается на серверную реализацию *HTTP*, чтобы передавать запросы приложению. Переданный запрос представляется как набор интерфейсов *feature*, которые приложение затем компонует в *HttpContext*. *ASP.NET Core* включает в себя кроссплатформенный веб сервер, *Kestrel*, который обычно запускается за производственным веб сервером, таким как *IIS* или *nginx*.

Корневая директория контента.

Корневая директория – это основной путь к любому контенту, используемому в приложении, например, его представления и веб контент. По

умолчанию корневая директория контента – это то же самое, что и основной путь приложения для выполняемого хостинга; альтернативу можно указать с помощью *WebHostBuilder*.

Корневая директория веб.

Корневая директория веб (*web root*) – это директория для открытых статических ресурсов, таких как файлов *css, js* и файлов изображений. Связующее ПО статических файлов по умолчанию отрабатывает файлы только из этой директории (и поддиректорий). Путем директории является *<content root>/wwwroot*, но вы можете указать и другой путь с помощью *WebHostBuilder*.

Конфигурация.

ASP.NET Core использует новую конфигурационную модель для обработки простых пар «имя-значение». Новая конфигурационная модель не основывается на *System.Configuration* или *web.config*; она создается от упорядоченного набора конфигурационных провайдеров. Встроенные конфигурационные провайдеры поддерживают множество форматов файлов (*XML, JSON, INI*) и переменных среды. Также можно написать собственные пользовательские конфигурационные провайдеры.

Среды.

Среды, как «*Development*» и «*Production*» в *ASP.NET Core* можно настроить с помощью переменных среды.

Создание веб *UI* и веб *API* с помощью *ASP.NET Core MVC*.

- Возможность создания хорошо тестируемого веб приложения с отличной архитектурой, которые следуют паттерну *Model-View-Controller (MVC)*.

- Возможность создания *HTTP* сервиса, который поддерживает несколько форматов.

- *Razor* предлагает язык для создания представлений.

- Тэг-хэлперы включают серверный код для создания и отображения *HTML* элементов в *Razor* файлах.

- Возможность создания *HTTP* сервиса с полной поддержкой согласования содержания, используя пользовательское или встроенное форматирование (*JSON, XML*).

- Связывание моделей автоматически передает данные из *HTTP* запросов параметрам методов действия.

- Валидация модели автоматически выполняет клиентскую и серверную валидацию.

- Разработка на стороне клиента.

- *ASP.NET Core* интегрируется со множеством клиентских фреймворков, включая *AngularJS, KnockoutJS* и *Bootstrap*.

1.6.3 Паттерн *MVC*

Одним из отличительных моментов платформы *ASP.NET Core* [18] является применение паттерна *MVC*. Причем последняя версия *MVC*-фреймворка, который применяется в *ASP.NET Core*, имеет номер 3.0/3.1. Поэтому важно не путать *ASP.NET MVC* 5, который применяется в *ASP.NET* 4.5-4.8, и фреймворк *MVC*, который применяется в *ASP.NET Core*. Хотя во многих аспектах эти фремйоворки будут совпадать.

Также неверно отождествлять *ASP.NET Core* всецело с фреймворком *ASP.NET Core MVC*. Фреймворк *ASP.NET Core MVC* работает поверх платформы *ASP.NET Core*, и предназначен для того, чтобы упростить создание приложения. Но мы можем и не использовать *MVC*, а применять чистый *ASP.NET Core* и на нем всецело выстраивать логику приложения.

Сам паттерн *MVC* не является какой-то новой идеей в архитектуре приложений, он появился еще в конце 1970-х годов в компании *Xerox* как способ организации компонентов в графическом приложение на языке *Smalltalk*.

Концепция паттерна *MVC* [18] предполагает разделение приложения на три компонента, схему взаимодействий которых можно увидеть на рисунке 12:

- модель (*model*): описывает используемые в приложении данные, а также логику, которая связана непосредственно с данными, например, логику валидации данных. Как правило, объекты моделей хранятся в базе данных.

В *MVC* модели представлены двумя основными типами: модели представлений, которые используются представлениями для отображения и передачи данных, и модели домена, которые описывают логику управления данными.

Модель может содержать данные, хранить логику управления этими данными. В то же время модель не должна содержать логику взаимодействия с пользователем и не должна определять механизм обработки запроса. Кроме того, модель не должна содержать логику отображения данных в представлении.

- представление (*view*): отвечают за визуальную часть или пользовательский интерфейс, нередко *html*-страница, через который пользователь взаимодействует с приложением. Также представление может содержать логику, связанную с отображением данных. В то же время представление не должно содержать логику обработки запроса пользователя или управления данными.

- контроллер (*controller*): представляет центральный компонент *MVC*, который обеспечивает связь между пользователем и приложением, представлением и хранилищем данных. Он содержит логику обработки запроса пользователя. Контроллер получает вводимые пользователем данные и обрабатывает их. И в зависимости от результатов обработки отправляет пользователю определенный вывод, например, в виде представления, наполненного данными моделей.

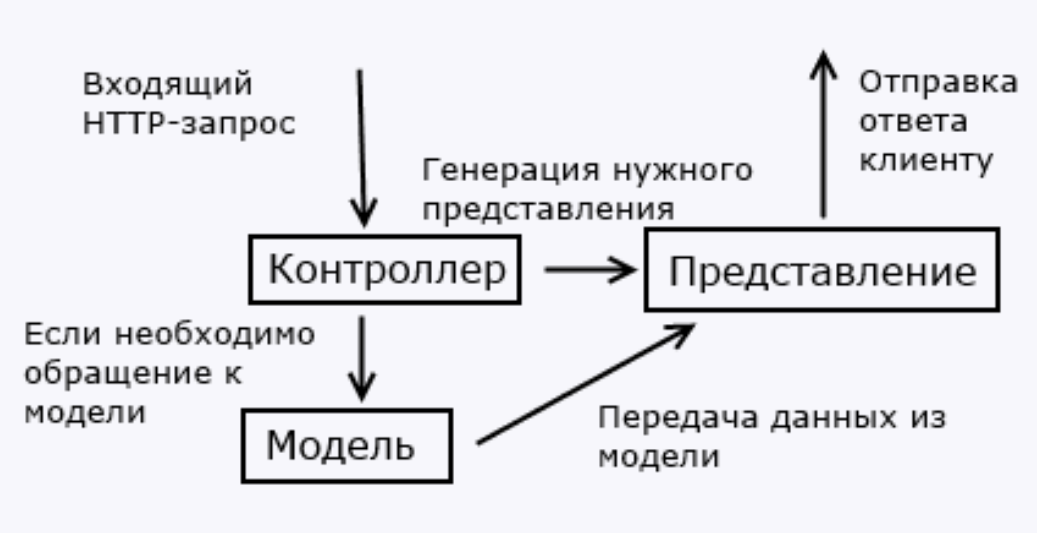

Рисунок 12 – Схема взаимосвязей между компонентами паттерна

В этой схеме [18] модель является независимым компонентом – любые изменения контроллера или представления никак не влияют на модель. Контроллер и представление являются относительно независимыми компонентами. Так, из представления можно обращаться к определенному контроллеру, а из контроллера генерировать представления, но при этом нередко их можно изменять независимо друг от друга.

Такое разграничение компонентов приложения позволяет реализовать концепцию разделение ответственности, при которой каждый компонент отвечает за свою строго очерченную сферу. В связи с чем легче построить работу над отдельными компонентами. И благодаря этому приложение легче разрабатывать, поддерживать и тестировать отдельные компоненты. Допустим, если важна визуальная часть или фронтэнд, то можно тестировать представление независимо от контроллера. Либо сосредоточиться на бэкэнде и тестировать контроллер.

Однако неверно полностью ассоциировать всю платформу *ASP.NET Core* с *MVC*. *MVC* – это лишь паттерн, который реализуется в рамках платформы. Есть возможность создать проект по шаблону *Empty*, где не будет никаких контроллеров, моделей, представлений, где будет один класс *Startup*. И через этот класс построить всю обработку запроса. Но естественно применение *MVC* облегчает разработку приложений.

Структура создаваемого проекта [18] будет отличаться от структуры проекта по типу *Empty*. В частности, будет виден ряд новых папок и файлов, которые изображены на рисунке 13.

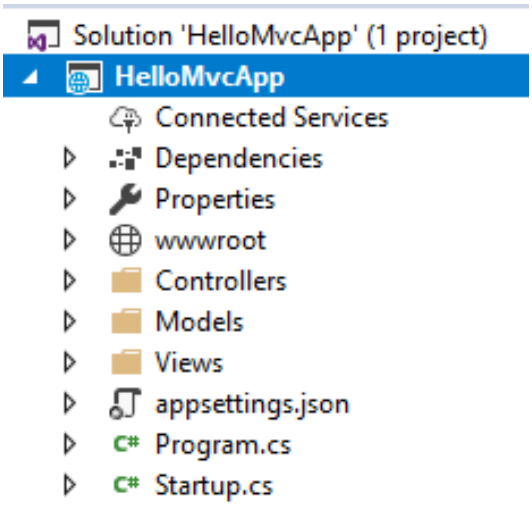

Рисунок 13 – Структура проекта с использованием паттерна

- *Dependencies*: все добавленные в проект пакеты и библиотеки;
- *wwwroot*: этот узел (на жестком диске ему соответствует одноименная папка) предназначен для хранения статических файлов - изображений, скриптов *javascript*, файлов *css* и так далее, которые используются приложением. Цель добавления этой папки в проект по сравнению с другими версиями *ASP.NET*, состоит в разграничении доступа к статическим файлам, к которым разрешен доступ со стороны клиента и к которым доступ запрещен;
- *Controllers*: папка для хранения контроллеров, используемых приложением;
- *Models*: каталог для хранения моделей;
- *Views*: каталог для хранения представлений;
- *appsettings.json*: хранит конфигурацию приложения;
- *Program.cs*: файл, определяющий класс *Program*, который инициализирует и запускает хост с приложением;
- *Startup.cs*: файл, определяющий класс *Startup*, с которого начинается работа приложения. То есть это входная точка в приложение.

Фактически эта та же структура, что и у проекта по типу *Empty* за тем исключением, что здесь также добавлены по умолчанию папки для ключевых компонентов фреймворка *MVC*: контроллеров и представлений. А также есть дополнительные узлы и файлы для управления зависимостями клиентской части приложения.

1.6.4 Технология *WebGL*

WebGL [18] представляет собой технологию, базирующуюся на *OpenGL ES* 2.0 и предназначенную для рисования и отображения интерактивной 2*D*- и 3*D*-графики в веб-браузерах. При этом для работы с данной технологией не требуются сторонние плагины или библиотеки. Вся работа веб-приложений с использованием *WebGL* основана на коде *JavaScript*, а некоторые элементы кода − шейдеры могут выполняться непосредственно на графических процессорах на видеокартах, благодаря чему разработчики могут получить доступ к дополнительным ресурсам компьютера, увеличить быстродействие. Таким образом, для создания приложений разработчики могу использовать стандартные для веб-среды технологии *HTML/CSS/JavaScript* и при этом также применять аппаратное ускорение графики.

Если создание настольных приложений, работающих с 2*D* и 3*D* − графикой нередко ограничивается целевой платформой, то здесь главным ограничением является только поддержка браузером технологии *WebGL*. А сами веб-приложения, построенные с использованием данной платформы, будут доступны в любой точке земного шара при наличии сети интернет вне зависимости от используемой платформы: то ли это десктопы с *ОС Windows*, *Linux, Mac*, то ли это смартфоны и планшеты, то ли это игровые консоли.

WebGL [18] возник из экспериментов над *Canvas* 3*D* американского разработчика сербского происхождения Владимира Вукичевича из компании *Mozilla* в 2006 году. Впоследствии разработчики браузеров *Opera* и *Mozilla* стали создавать свои реализации *WebGL*. А впоследствии была организована

рабочая группа с участием крупнейших разработчиков браузеров *Apple*, *Google, Mozilla, Opera* для работы над спецификацией технологии. И в 3 марта 2011 года была представлена спецификация *WebGL* 1.0.

Поддержка браузерами.

В настоящий момент *WebGL* поддерживается следующими браузерами:

Десктопные браузеры:

- *Mozilla Firefox* (с 4-й версии);
- *Google Chrome* (с 9-й версии);
- *Safari* (с 6-й версии, по умолчанию поддержка *WebGL* отключена);
- *Opera* (с 12-й версии, по умолчанию поддержка *WebGL* отключена);
- *IE* (с 11-й версии, для других версий можно воспользоваться сторонними плагинами, например, *IEWebGL*).

Мобильные браузеры и платформы:

- *Android*-браузер (поддерживает *WebGL* только на некоторых устройствах, например, на смартфонах *Sony Ericsson Xperia* и некоторых смартфонах *Samsung*)
- *Opera Mobile* (начинаю с 12-й версии и только для *ОС Android*);
- *IOS;*
- *Firefox for mobile;*
- *Google Chrome* для *Android.*

Преимущества использования *WebGL:*

- кроссбраузерность и отсутствие привязки к определенной платформе. *Windows, MacOS, Linux* - все это не важно, главное, чтобы браузер поддерживал *WebGL;*
- использование языка *JavaScript*, который достаточно распространен;
- автоматическое управление памятью. В отличие от *OpenGL* в *WebGL* не надо выполнять специальные действия для выделения и очистки памяти;
- поскольку *WebGL* для рендеринга графики использует графический процессор на видеокарте (*GPU*), то для этой технологии характерна высокая

производительность, которая сравнима с производительностью нативных приложений.

На рисунке 14 можно увидеть подробную схему графического конвейера, где для рендеринга 3*D* графики следует выполнить последовательность шагов.

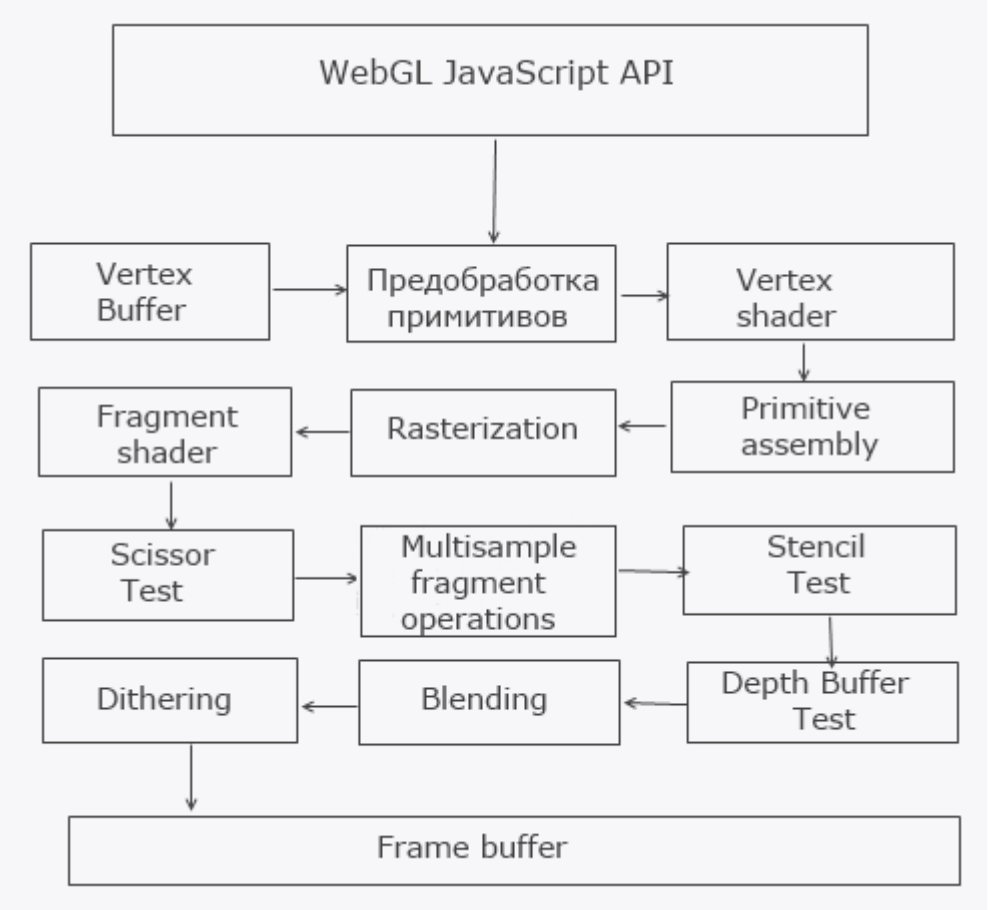

Рисунок 14 – Схема конвейера *WebGL*

Теперь поэтапно.

1. Вначале мы создаем набор вершин в буфере вершин (*Vertex Buffer*). По этим вершинам впоследствии будут составлены геометрические примитивы, а из примитивов – объекты. И проводим некоторую предобработку.

2. Затем содержимое буфера вершин поступает на обработку в вершинный шейдер (*Vertex Shader*). Шейдер производит над вершинами некоторые трансформации, например, применяет матрицы преобразования и

так далее. Шейдеры пишутся самим разработчиком, поэтому программист может применить различные преобразования по своему усмотрению.

3. На следующем этапе (*Primitive Assembly*) конвейер получает результат вершинного шейдера и пытается измененные вершины сопоставить в отдельные примитивы - линии, треугольники, спрайты. Также на этом этапе определяется, входит ли примитив в видимое пространство. Если нет, то он обрезается. Оставшиеся примитивы передаются на следующий этап конвейера.

4. Далее на этапе растеризации (*Rasterization*) полученные примитивы преобразуются в фрагменты, которые можно представить как пиксели, которые затем будут отрисованы на экране

5. И затем в дело вступает фрагментный шейдер (*Fragment shader*). (В технологиях *Direct3D, XNA* прямым аналогом является пиксельный шейдер). Фрагментый шейдер производит преобразования с цветовой составляющей примитивов, наполняет их цветом, точнее окрашивает пиксели, и в качестве вывода передает на следующий этап измененные фрагменты.

Следующий этап [18] представляет собой ряд преобразований над полученными с фрагментного шейдера фрагментами. Собственно, состоит из нескольких подэтапов:

1. *Scissor Test*: на этом этапе проверяется, находится ли фрагмент в пределах отсекающего прямоугольника. Если фрагмент находится в пределах этого прямоугольника, то он передается на следующий этап. Если же нет, то он отбрасывается и больше не принимает участия в обработке.

2. *Multisample Fragment Operations*: на данном этапе у каждого фрагмента изменяются цветовые составляющие, производится сглаживание (*anti-alising*), чтобы объект выглядел более плавно на экране.

3. *Stencil Test*: здесь фрагмент передается в буфер трафаретов (*stencil buffer*). Если вкратце, то в этом буфере дополнительно отбрасываются те фрагменты, которые не должны отображаться на экране. Как правило, данный

буфер используется для создания различного рода эффектов, например, эффект теней.

4. *Depth Buffer Test* – тест буфера глубины. В буфере глубины (*depth buffer*, а также называется, *z-buffer*) сравнивается *z*-компонента фрагмента, и если она больше значения в буфере глубины, то, следовательно, данный фрагмент расположен к смотрящему на трехмерную сцену ближе, чем предыдущий фрагмент, поэтому текущий фрагмент проходит тест. Если же *z*-компонента больше значения в буфере глубины, то, следовательно, данный фрагмент находится дальше, поэтому он не должен быть виден и отбрасывается.

5. *Blending*: на данном этапе происходите небольшое смешение цветов, например, для создания прозрачных объектов.

6. *Dithering*: здесь происходит смешение цветов, для создания тонов и полутонов.

6. *Frame Buffer*: и здесь наконец полученные после предобработки фрагменты превращаются в пиксели на экране.

Итак, были рассмотрены этапы конвейера. Данные этапы рисуют некоторый алгоритм действий, от которого затем можно будет отталкиваться. И создание программы [18] разбивается также на некоторые этапы:

1. создание и настройка шейдеров;

2. создание и настройка буфера вершин, которые в последствии образуют геометрическую фигуру;

3. отрисовка фигуры.

1.6.5 Библиотека *three.js*

Three.js [23] – легковесная кроссбраузерная библиотека *JavaScript*, используемая для создания и отображения анимированной компьютерной 3*D* графики при разработке веб-приложений. *Three.js* скрипты могут

использоваться совместно с элементом *HTML5 CANVAS, SVG* или *WebGL*. Исходный код расположен в репозитории *GitHub*.

Обзор [23] .

Three.js позволяет создавать ускоренную на *GPU 3D* графику, используя язык *JavaScript* как часть сайта без подключения проприетарных плагинов для браузера. Это возможно благодаря использованию технологии *WebGL*. Поддерживает трёхмерные модели формата *Collada*.

Особенности [23]:

- рендереры: *Canvas, SVG* или *WebGL;*
- сцена: добавление и удаление объектов в режиме реального времени; туман;
- камеры: перспективная или ортографическая;
- анимация: каркасы, быстрая кинематика, обратная кинематика, покадровая анимация;
- источники света: внешний, направленный, точечный; тени: брошенные и полученные;
- шейдеры: полный доступ ко всем *OpenGL* шейдерам (*GLSL*);
- объекты: сети, частицы, спрайты, линии, скелетная анимация и другое;
- геометрия: плоскость, куб, сфера, тор, 3*D* текст и другое; модификаторы: ткань, выдавливание;
- загрузчики данных: двоичный, изображения, *JSON* и сцена;
- экспорт и импорт: утилиты, создающие Three.js-совместимые *JSON* файлы из форматов: *Blender*, *openCTM*, *FBX*, *3D Studio Max*, и *Wavefront .obj* файл;
- поддержка: документация по *API* библиотеки находится в процессе постоянного расширения и дополнения, есть публичный форум и обширное сообщество;

- примеры: на официальном сайте можно найти более 150 примеров работы со шрифтами, моделями, текстурами, звуком и другими элементами сцены.

Библиотека *Three.js* работает во всех браузерах, которые поддерживают технологию *WebGL*; также может работать с «чистым» интерфейсом элемента *CANVAS*, благодаря чему работает и на многих мобильных устройствах. *Three.js* распространяется под лицензией *MIT license*.

Основные понятия, схема взаимосвязи которых представлена на рисунке 15.

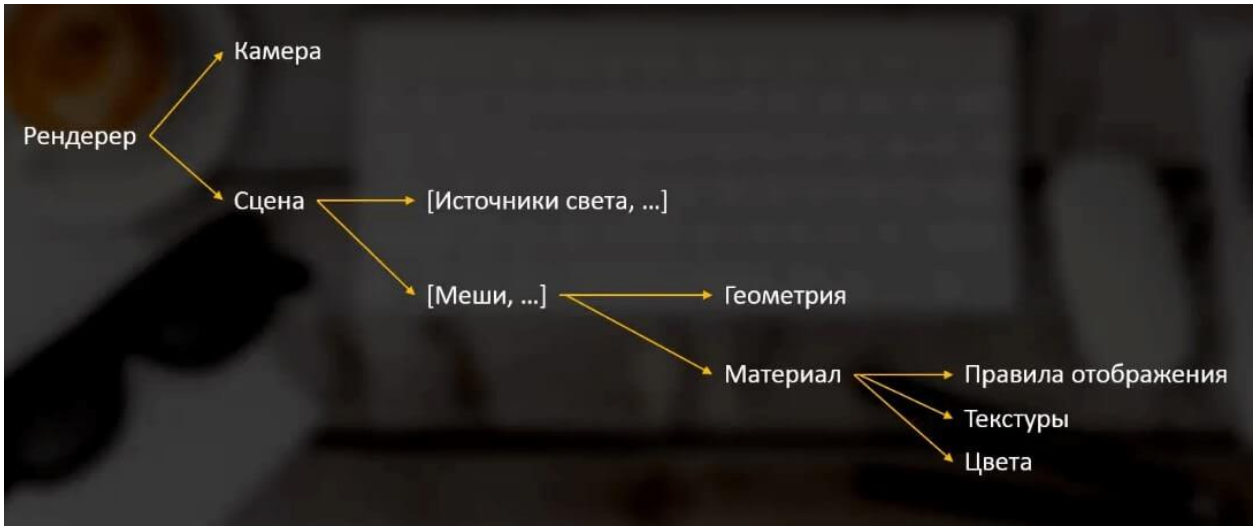

Рисунок 15 – Иерархия отображения

Первоначально есть рендерер – то, что работает с тегом *canvas* и выводит туда изображения. Он зависит от камеры и сцены: камера направлена на сцену. На сцене есть источники света – это массив, потому что их может быть много; много может быть и мешей – 3*D* объектов, которые заводятся на сцену. Каждый меш представляет собой геометрию и материал. Материал – это наиболее обширное понятие, есть «хитрые» материалы (например, с отражением). Но так или иначе, материал включает в себя правила отображения, возможность поставить цвета, текстуры и многое другое [17].

1.6.6 Математическая статистика. Многоугольник (полигон) распределения и гистограмма

Математическая статистика [10] – раздел математики, изучающий математические методы сбора, систематизации, обработки и интерпретации результатов наблюдений с целью выявления статистических закономерностей.

Генеральной совокупностью называется вся исследуемая совокупность объектов.

Она может состоять из конечного числа случайной величины, часто весьма ограниченного; в других случаях генеральная совокупность может быть бесконечной.

Чаще всего имеем дело с ситуацией [10], когда измерение каждого элемента генеральной совокупности невозможно из-за большого их числа. В таких случаях для характеристики генеральной совокупности прибегают к выборке из нее единиц наблюдения, и обработке подвергается лишь часть единиц изучаемой совокупности.

Выборочной совокупностью или просто выборкой называют совокупность случайно отобранных объектов.

Выборочным методом называется метод, основанный на том, что заключение о генеральной совокупности делается по результатам выборки.

Для применения выборочного метода необходимо, чтобы выборка правильно представляла свойства генеральной совокупности, или, говорят, что выборка должна быть репрезентативной (представительной). А это возможно, если все объекты генеральной совокупности имеют одинаковую вероятность (возможность) попасть в выборку.

Пусть из генеральной совокупности извлечена выборка, причем значение x_1 наблюдалось n_1 раз, x_2 наблюдалось n_2 раз, …, $x_к$ наблюдалось $n_к$ раз. Общий объем выборки можно определить как

$$n = \sum_{i=1}^{k} n_i.$$

Наблюдаемые значения x_i называют вариантами, а последовательность вариант, записанных в возрастающем порядке, – вариационным рядом.

Числа наблюдений называют частотами, а их отношения к объему выборки $\frac{n_i}{n} = W_i$ – относительными частотами.

Статистическим распределением выборки [10] называют перечень вариантов и соответствующих им частот или относительных частот. Статистическое распределение можно задать также в виде последовательности интервалов и соответствующих им частот (в качестве частоты, соответствующей интервалу, принимают сумму частот, попавших в этот интервал).

В теории вероятностей под распределением понимают соответствие между возможными значениями случайной величины и их вероятностями, а в математической статистике – соответствие между наблюдаемыми вариантами и их частотами, или относительными частотами.

Эмпирической функцией распределения [11] (функцией распределения выборки) называют функцию $F^*(X)$, определяющую для каждого значения x относительную частоту события $X < x$.

Итак, по определению,

$$F^*(X) = n_x/n,$$

где n_x – число вариант, меньших x;

n – объем выборки.

Необходимость выборочного метода вызвана тем, что объект исследования очень обширный (весь материал, находящийся на ленточном конвейере), и к тому же не представляется возможным рассмотреть каждую

составляющего это объекта, потому что измерительный комплекс только предоставляет верхние узловые точки с заданным шагом. Следовательно формирование выборки достигается за счет метода случайного отбора.

Так как главной задачей выборочного обследования является получение с минимальным объемом выборки как можно более точного описания интересующей генеральной совокупности на основе выборочных данных, этот метод отлично подходит для применения расчетов фракционности и уточнения вычислений объема технического материала.

Соответственно, в случае сыпучего сырья, где мы знаем только его насыпную кривую, генеральная совокупность будет представлять из себя все зерна предоставленного материала, какими бы они не были по размеру фракций и форм для групп лещадности. А выборкой будут являться те зерна, которые будут измеряться для определения параметров фракционности и лещадности, а именно такие, которые алгоритм выберет для проверки (выбранные экземпляры находятся сверху всей насыпной кучи, причем не загорожены другими зернами и не утоплены вглубь, так как это позволяет исключить большое количество ошибок, другими словами, подбираются экземпляры, которые явно лежат на поверхности всем своим телом).

Двумерная случайная величина (X; Y) дискретна [11], если X, Y – дискретные случайные величины. Для нее составляется таблица распределения – аналог ряда распределения для одномерной случайной величины.

Закон распределения дискретной двумерной случайной величины (X;Y) имеет вид таблицы с двойным входом, задающей перечень возможных значений каждой компоненты и вероятности $p(xi, yj)$, с которыми величина принимает значение (xi, yj).

Таблица 6 – Распределение дискретной двумерной случайной величины

X	Y					
	y_1	y_2	…	y_j	…	y_n
x_1	$p(x_1, y_1)$	$p(x_1, y_2)$	…	$p(x_1, y_j)$	…	$p(x_1, y_n)$

…	…	…	…	…	…	…
x_i	$p(x_i, y_1)$	$p(x_i, y_2)$	…	$p(x_i, y_j)$	…	$p(x_i, y_n)$
…	…	…	…	…	…	…
x_m	$p(x_m, y_1)$	$p(x_m, y_2)$	…	$p(x_m, y_j)$	…	$p(x_m, y_n)$

Именно в такую таблицу записываются рассчитанные данные фракционности и лещадности (которая представляет собой двумерный массив). Сумма всех полученных вероятностей составляет 1, либо, если вместо частоты используется относительная частота, то сумма вероятностей будет равна 100% [12].

Наглядное представление о статистическом распределении выборки дают графики.

Полигоном частот [12] называют ломаную, отрезки которой соединяют точки $(x_1; n_1,)$, $(x_2; n_2),..., (x_к; n_к)$. Для построения полигона частот на оси абсцисс откладывают варианты x, а на оси ординат – соответствующие им частоты n_i. Точки соединяют отрезками прямых и получают полигон частот.

Полигоном относительных частот называют ломаную (рисунок 16), отрезки которой соединяют точки $(x_1; W_1), (x_2\ W_2), ..., (x_k; W_k)$. Для построения полигона относительных частот на оси абсцисс откладывают варианты x, а на оси ординат – соответствующие им относительные частоты W_i. Точки соединяют отрезками прямых и получают полигон относительных частот.

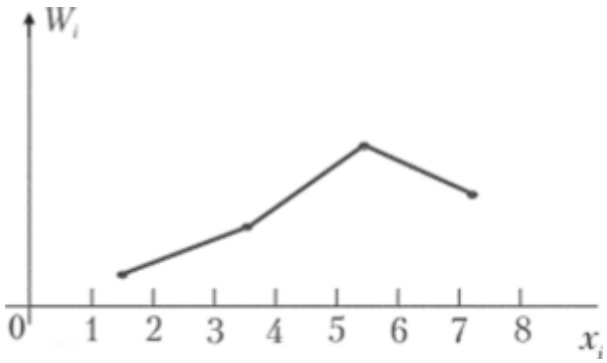

Рисунок 16 - Пример полигона относительных частот

В случае непрерывного признака [11] целесообразно строить гистограмму, для чего интервал, в котором заключены все наблюдаемые

значения признака, разбивают на несколько частичных интервалов длиной h и находят для каждого частичного интервала n_i – сумму частот вариант, попавших в i-й интервал.

На рисунке 17 изображена гистограмма частот – ступенчатая фигура, состоящая из прямоугольников, основаниями которых служат частичные интервалы, а высоты равны отношению n_i/h (плотность частоты).

Для построения гистограммы частот на оси абсцисс откладывают частичные интервалы, а над ними проводят отрезки, параллельные оси абсцисс на расстоянии n_i/h.

Площадь гистограммы частот равна сумме всех частот, то есть объему выборки.

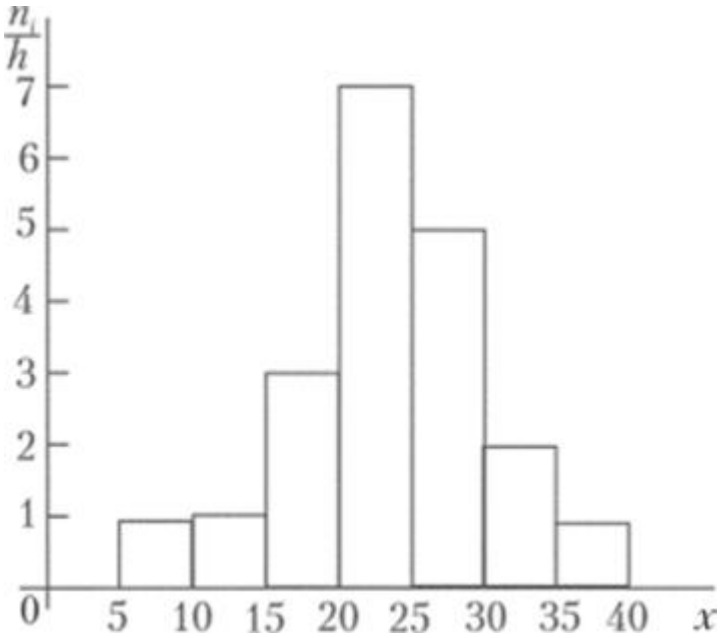

Рисунок 17 – Пример гистограммы частот распределения

Гистограммой относительных частот [11] называют ступенчатую фигуру, состоящую из прямоугольников, основаниями которых служат частичные интервалы длиною A, а высоты равны отношению W_i/h (плотность относительной частоты).

Для построения гистограммы относительных частот на оси абсцисс откладывают частичные интервалы, а над ними проводят отрезки, параллельные оси абсцисс на расстоянии W_i/h.

Площадь гистограммы относительных частот равна сумме всех относительных частот, то есть единице.

Следовательно, при построении графиков одна из осей (OY) будет представлять собой относительную частоту, которая измеряется в процентах, а две другие оси для многомерной гистограммы, или двумерных графиков будут характеристики сыпучего материала, а именно фракционность и отношение сторон (лещадность).

Для построения гистограммы [12] в прямоугольной системе координат на оси абсцисс откладываем интервалы, а на каждом из интервалов h_i, как на основании, строим прямоугольник с основанием h_i и высотой $f(x_i) = \frac{v_i}{n \times h_i}$. Полученная таким образом фигура и называется гистограммой выборки или эмпирической плотностью распределения.

На этом же графике строим гипотетическую (предположительную) теоретическую нормальную кривую.

Для этого берем середины интервалов x_i и выравнивающие частоты v_i наносим точки с координатами $(x_i, \frac{v_i'}{nh})$, соединяя их плавной линией, получаем искомую кривую [13].

1.7 Постановка задачи

После подробного изучения теоретического материала, связанного с измерением объема технического сырья, характеристиками и основными свойствами материалов, проведенным анализом алгоритмов математического обеспечения существующих измерительных комплексов была поставлена задача. А именно, расширить функционал алгоритма, учитывая фракционность зерен материала и лещадность форм сырья. Полученные данные предоставить пользователю в виде гистограммы.

Помимо расширения функционала, следует произвести расчеты количества объема измеряемого насыпного сырья. И не только наиболее точно

высчитать это значение, но и создать визуализацию, по которой можно будет увидеть 3*D* модель конвейера и 2*D* сечение с насыпной линией, узловые точки которой предоставил лазерный измерительный комплекс.

2 Разработка алгоритма расчета объема технического сырья

Алгоритм расчета объема материала, без воздушных прослоек, представленный на рисунке 18, состоит из трех обширных шагов, каждый из которых подробно будет рассматривать далее.

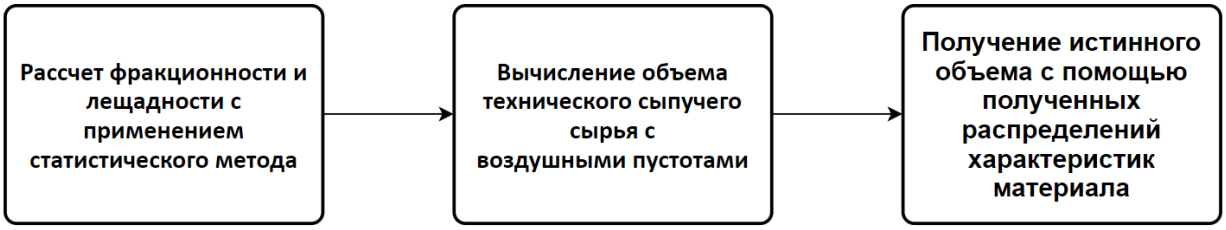

Рисунок 18 – Блок-схема краткого алгоритма для достижения поставленной задачи

2.1 Определение фракционного состава сырья по размерам, лещадности

Фракционность – условное разделение материала на виды по размеру его зерна. Размер зерен может быть разным. Существуют основные фракции, которые упоминаются в соответствующих нормативных документах (ГОСТ). Но при производстве данных фракций часто получается материал, который не соответствует правилам и образует сопутствующие фракции.

В результате дробления горной породы образуется определенная часть камней с отклоняющимися от нормы параметрами. В общей массе появляются материалы игольчатой, плоской форм. Это явление называют лещадностью. Щебень с кубической или остроугольной (сферической) формой отлично

утрамбовывается, что обеспечивает надежную прочность будущим сооружениям. Игольчатые и пластинчатые элементы плохо прилегают друг к другу, к тому же они хрупкие и быстро ломаются от нагрузок.

Лещадность щебня показывает количество стандартных компонентов в составе стройматериала. Прочный камень отличается низким показателем доли дефекта.

Зерна, которые имеют длину в три раза больше толщины или ширины, будут считаться пластинчатыми и игловатыми. Пример таких форм видно на рисунке 19.

Рисунок 19 – Определение лещадности щебня

Именно из-за этого и нужны расчеты для более точного измерения объема материала, благодаря которым, предоставляется возможность для каждой фракции использовать свою насыпную плотность и оценивать качество насыпного материала по его лещадности. Алгоритм представлен на рисунке 20.

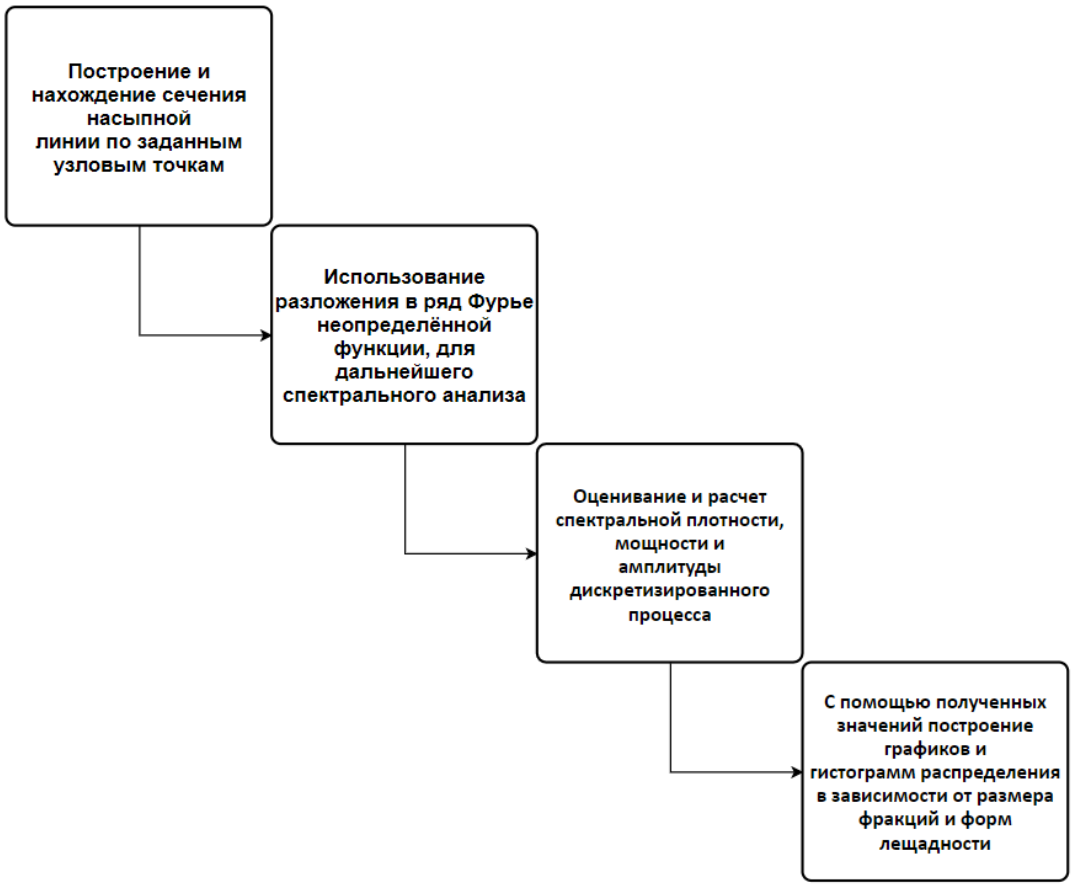

Рисунок 20 – Блок-схема алгоритма расчета фракционности и лещадности с применением статистического метода и спектрального анализа

Для разбиения по фракциям всего технического сырья, которое находится на ленточном конвейере, необходимо проводить анализ нескольких сечений (рисунок 21), следующих друг за другом, так как одна из осей представляет шаг движения конвейера (*Oz*).

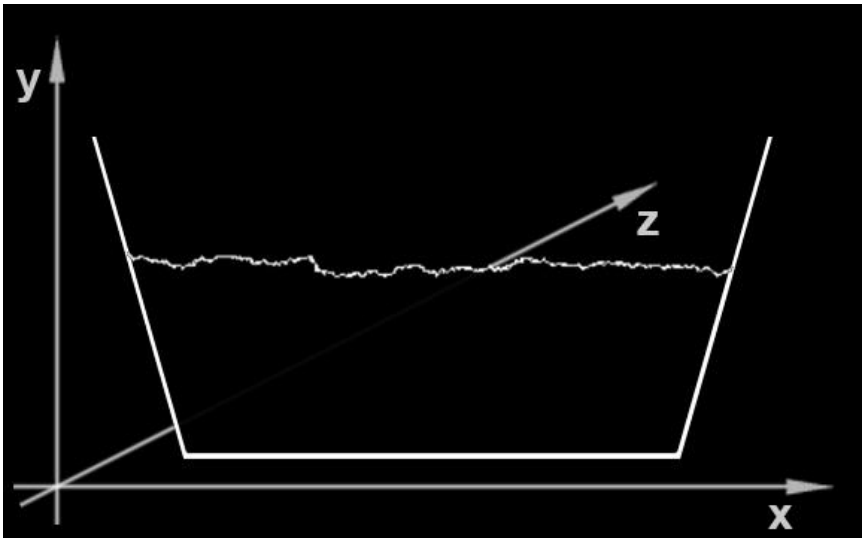

Рисунок 21 – Сечение ленточного конвейера в системе координат

На рисунке можно увидеть, что насыпная кривая имеет незначительные колебания, которые скорее всего связаны с пористостью, изгибами или неровностями технического сырья. Такие колебания в расчет разбиения по фракциям не учитываются.

Так же невооруженным взглядом видно, что в этой кривой прослеживаются отдельные зерна технического сыпучего материала, но для точного обнаружения следует просматривать и следующие сечения по этим же значениям осей.

В начале, в первом сечении следует найти участки кривой, где ее значения создают некую гладкость, то есть на отрезке от 5 и выше не имеется сильных и частых колебаний, и учесть его отмеренное значение, пример такого участка изображен на рисунке 22. Для этого следует измерять угол между двумя отрезками с общей точкой, на протяжении всего выбранного участка, так как среднеарифметическое значение этих углов будет служить хорошей проверкой на его гладкость.

Рисунок 22 – Увеличенная кривая насыпной линии

Далее, в следующих сечениях, помимо повторения первого этапа, значение выбранных участков сравнивается и в этих сечениях. Если они сильно не разнятся, то продвигаться по оси *Oz* до тех пор, пока не появятся резкие скачки, или колебания, не соответствующие прошлым сечениям.

Следует учесть, что зерно может лежать под любым углом наклона к любой из осей, и нужно опираться не только на идентичные величины осей, но и на схожесть выбранного гладкого участка прошлого сечения, учитывая его незначительный сдвиг по осям координат.

После производится расчет количества пройденных сечений по оси *Oz* для сравнения с значением ребра полученном в начальном этапе. Та величина, которая больше и будет являться фракционностью данного зерна. И для определения группы лещадности зерна материала следует ребро, которое длиннее, разделить на второе ребро. Полученный результат сравнивать с числом 3, так как именно этот показатель даст искомую информацию.

И все-таки для построения распределения следует воспользоваться спектральным анализом [14], математической основой которого является преобразование Фурье, связывающее временное и частотное представления сигналов. Важную роль в спектральном анализе играют методы статистики, так как анализируемые сигналы, как правило, являются случайными.

Спектральный метод позволяет выделить из сложного сигнала более простые составляющие и определить, каковы их интенсивности. В качестве меры интенсивности сигнала и его простейших составляющих принято использовать мощность. Основное преимущество использования мощности в качестве меры интенсивности заключается в том, что для мощности сложного колебания, а не для амплитуды, справедливо свойство аддитивности: мощность любого сложного сигнала равна сумме мощностей его отдельных составляющих. Основной целью спектрального анализа являются оценивание спектральной плотности мощности дискретизированного случайного процесса и обнаружение в нем периодических составляющих.

Другими словами, спектральный анализ изучает спектр частоты, содержавшийся в дискретном, однородно выборочных данных. Преобразование Фурье является инструментом, который показывает частотные составляющие времени - или основанный на пробеле сигнал путем представления его в пространстве частоты, что видно на рисунке 23.

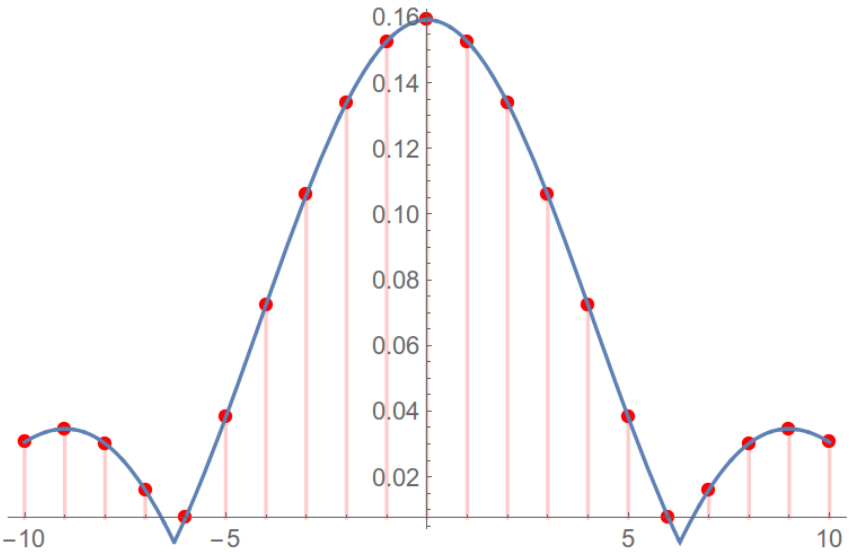

Рисунок 23 – Соответствие ряда Фурье и преобразования Фурье на примере амплитудного спектра

Так как имеем непериодическую функцию, то и разложение в ряд Фурье будет выглядеть следующим образом.

Разложение в ряд Фурье в интервале $[-L,L]$.

Рассмотрим кусочно-непрерывную $f(x)$, заданную в интервале $[-L,L]$. Используя подстановку $x=(Ly)/\pi$ $(-\pi \le x \le \pi)$, преобразуем ее в функцию

$$F(y) = f(\frac{Ly}{\pi}),$$

определенную и интегрируемую в интервале $[-\pi,\pi]$. Разложение в ряд Фурье для функции $F(y)$ имеет вид

$$F(y) = f\left(\frac{Ly}{\pi}\right) = \frac{a_0}{2} + \sum_{n=1}^{\infty}(a_n \cos ny + b_n \sin ny).$$

Коэффициенты Фурье для данной функции определяются формулами

$$a_0 = \frac{1}{\pi}\int_{-\pi}^{\pi} F(y)dy,$$

$$a_n = \frac{1}{\pi}\int_{-\pi}^{\pi} F(y)\cos ny\,dy = \frac{1}{\pi}\int_{-\pi}^{\pi} f\left(\frac{Ly}{\pi}\right)\cos ny\,dy\,,$$

$$b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} F(y)\sin ny\,dy = \frac{1}{\pi}\int_{-\pi}^{\pi} f\left(\frac{Ly}{\pi}\right)\sin ny\,dy\,, n = 1,2,3,\ldots$$

Возвращаясь к первоначальным переменным, то есть полагая $y=(\pi x)/L$, получим следующие выражения для ряда Фурье исходной функции $f(x)$:

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty}\left(a_n \cos\frac{n\pi x}{L} + b_n \sin\frac{n\pi x}{L}\right),$$

где

$$a_0 = \frac{1}{L}\int_{-L}^{L} f(x)\,dx,$$

$$a_n = \frac{1}{L}\int_{-L}^{L} f(x)\cos\frac{n\pi x}{L}\,dx,$$

$$b_n = \frac{1}{L}\int_{-L}^{L} f(x)\sin\frac{n\pi x}{L}\,dx.$$

Разложение в ряд Фурье в интервале $[a,b]$.

Если функция $f(x)$ определена в интервале $[a,b]$, то ее разложение в ряд Фурье определяется той же самой формулой

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty}\left(a_n \cos\frac{n\pi x}{L} + b_n \sin\frac{n\pi x}{L}\right),$$

где $L=(b-a)/2$, а коэффициенты вычисляются следующим образом:

$$a_0 = \frac{1}{L}\int_{-L}^{L} f(x)\,dx,$$

$$a_n = \frac{1}{L}\int_{-L}^{L} f(x)\cos\frac{n\pi x}{L}\,dx,$$

$$b_n = \frac{1}{L}\int_{-L}^{L} f(x)\sin\frac{n\pi x}{L}\,dx\,,\ \mathrm{n} = 1,2,3,\ldots$$

Четные и нечетные функции (рисунок 24).

Разложение в ряд Фурье четной функции, определенной в интервале $[-L,L]$, имеет вид

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos\frac{n\pi x}{L}$$

где

$$a_0 = \frac{2}{L}\int_{0}^{L} f(x)\,dx,$$

$$a_n = \frac{2}{L}\int_{0}^{L} f(x)\cos\frac{n\pi x}{L}\,dx.$$

Разложение в ряд Фурье нечетной функции, заданной в интервале $[-L,L]$, выражается формулой

$$f(x) = \sum_{n=1}^{\infty} b_n \sin\frac{n\pi x}{L},$$

где коэффициенты Фурье равны

$$b_n = \frac{1}{L}\int_{-L}^{L} f(x)\sin\frac{n\pi x}{L}\,dx$$

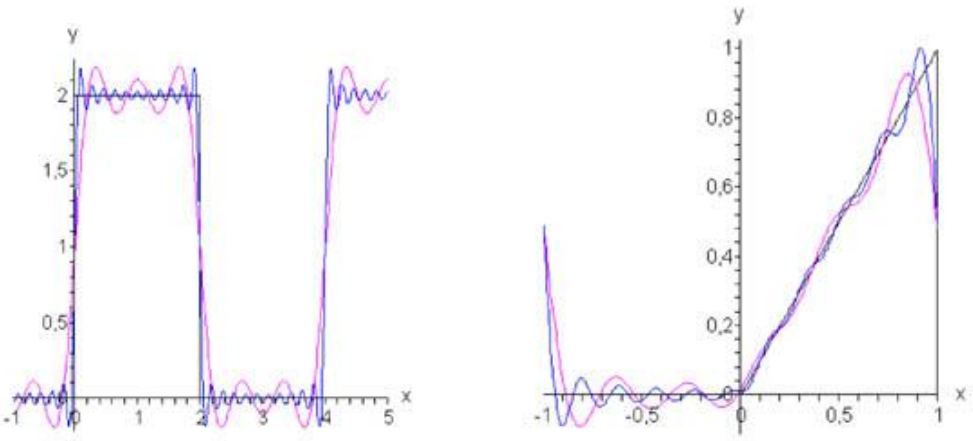

Рисунок 24 – Пример разложения ряда Фурье, где на первом графике $A = 2$, $L = 2$, $n = 2$, $n = 10$, а на втором $n = 5$, $n = 10$

В дальнейшем, предполагая, что содержание разных фракций зерен и их форм по лещадности технического сыпучего материала внутри конвейера такое же, что и на поверхности, все полученные значения и количество их совпадений запоминается для процентного соотношения.

Благодаря произведенным расчетам, зная процентное содержание определённых фракций, используется разная насыпная плотность, что позволяет более точно рассчитать объем сыпучего технического сырья, а опираясь на полученные данные по лещадности материала, появляется возможность отнести его к одной из групп, и провести анализ качества сырья.

На рисунках 25 – 28 изображены примеры графиков гистограмм и кривых расчета распределения различных фракционности зерен сырья. Также на рисунке 29 можно увидеть симметричность и асимметричность кривой распределения фракционного состава.

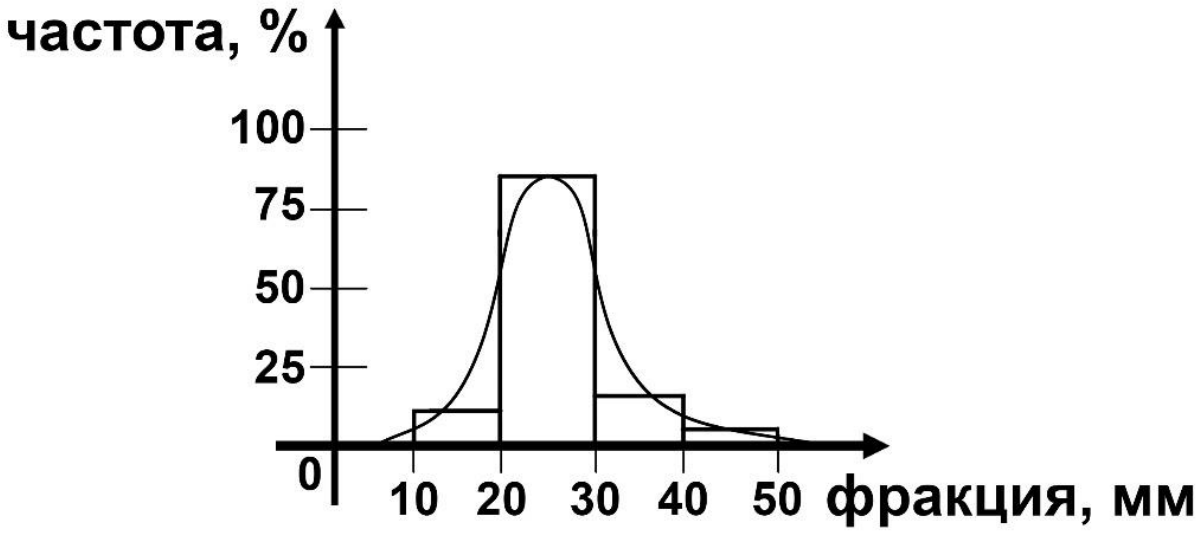

Рисунок 25 – Хорошо сортированная порода (преобладает фракция 20-30)

В таком случае сырье ленточного конвейера будет приближенно к однородному, что хорошо повлияет на дальнейшие работы с ним.

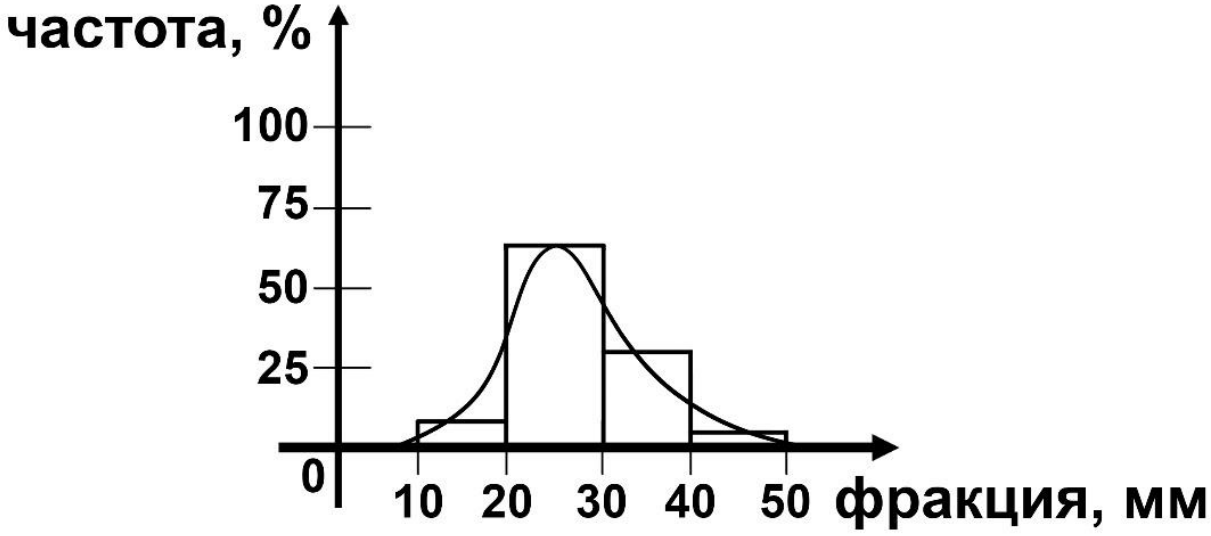

Рисунок 26 – Среднесортированное сырье (примесь фракции 30-40 к основной 20-30)

На рисунке 26 видно, что материал среднесортирован, есть явное преобладание одного размера фракции, но разница между всеми небольшая.

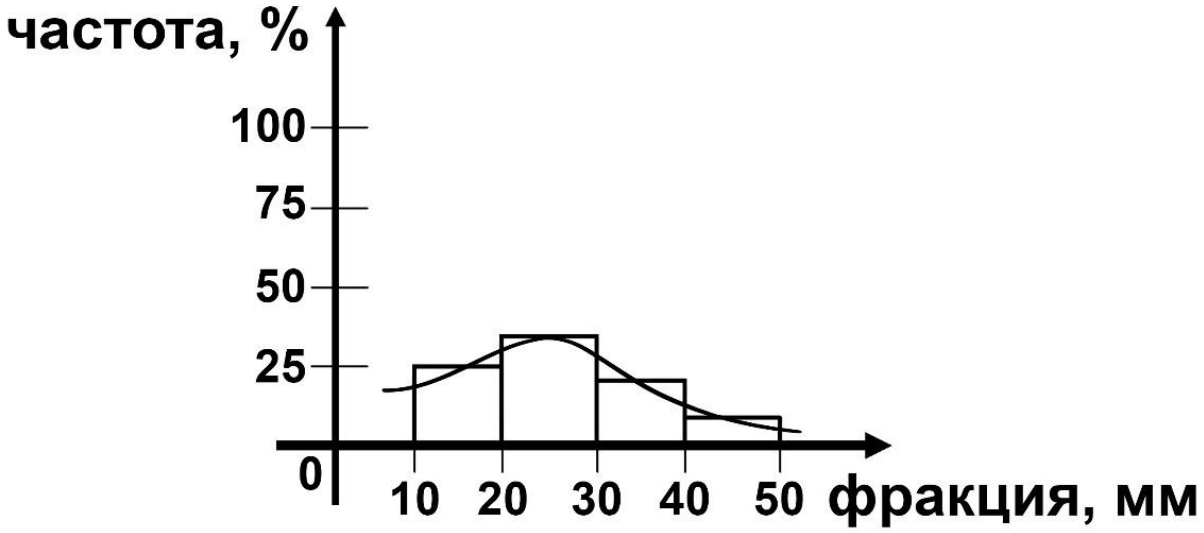

Рисунок 27 – Плохо сортированный материал

В данном случае нет преобладания ни одной фракции, это может быть связано с породой материала, с дроблением при транспортировке сырья.

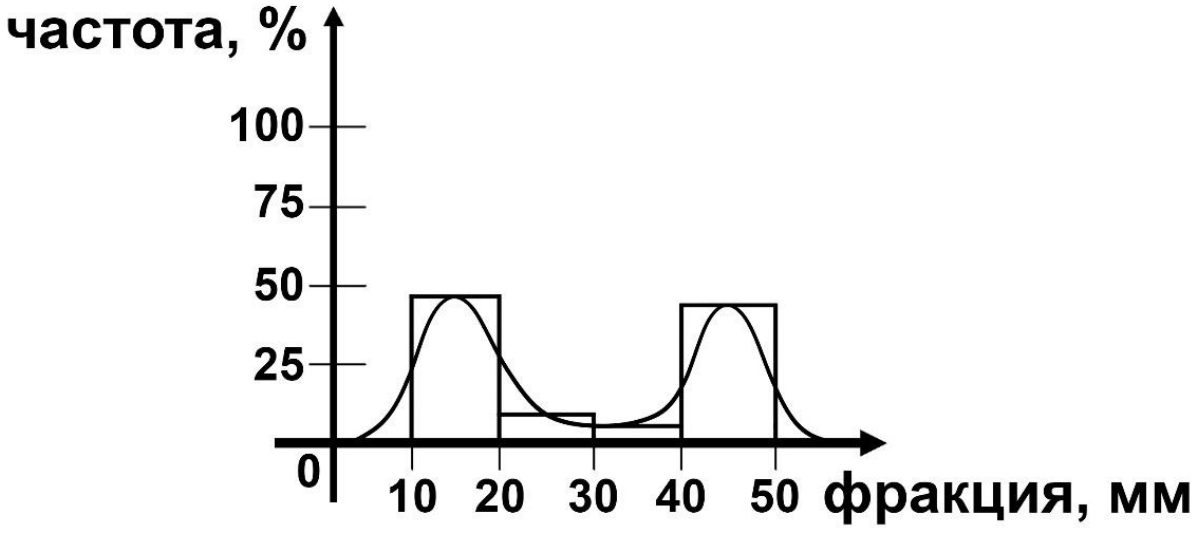

Рисунок 28 – Переслаивание пород (в данном случае фракций 10-20 и 40-50)

Такая ситуация, которая представлена на рисунке 28 может быть вызвана ошибкой поставщика, связанная с человеческим фактором или же сбоем при погрузке.

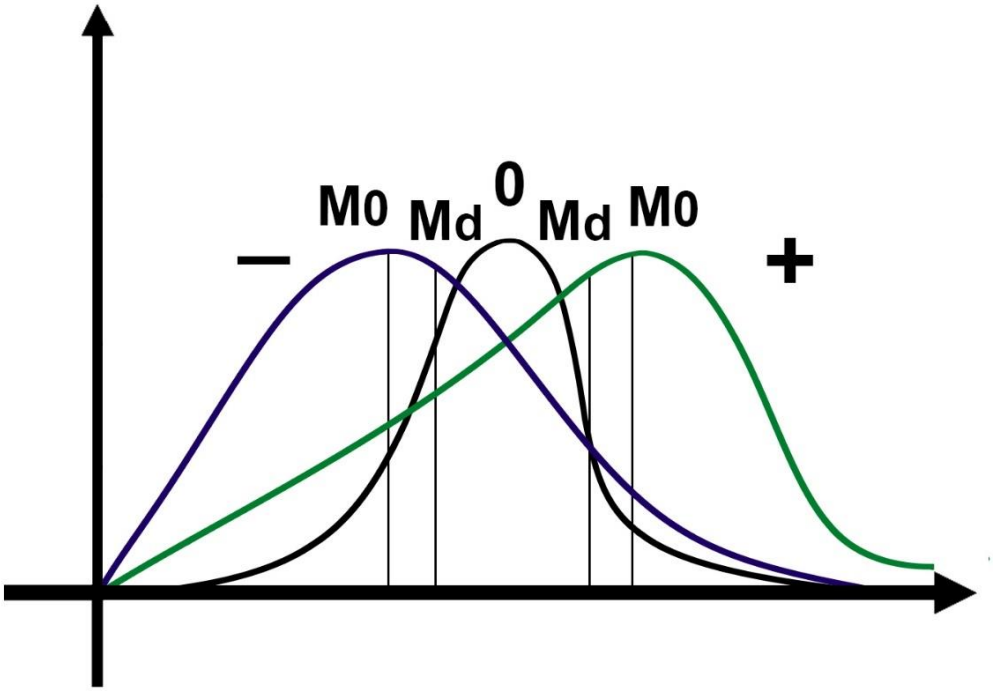

Рисунок 29 – Кривые распределения фракционного состава. Симметричная (0); ассиметричная с положительной (+) и отрицательной асимметрией (-). M_0 – мода, M_d – медиана

В графиках, которые изображены ниже, а именно рисунки 30, 31, 32, показана зависимость частоты к отношению длин ребер, так как именно эта зависимость определяет лещадность, а именно процент дефекта зерна к общему объему.

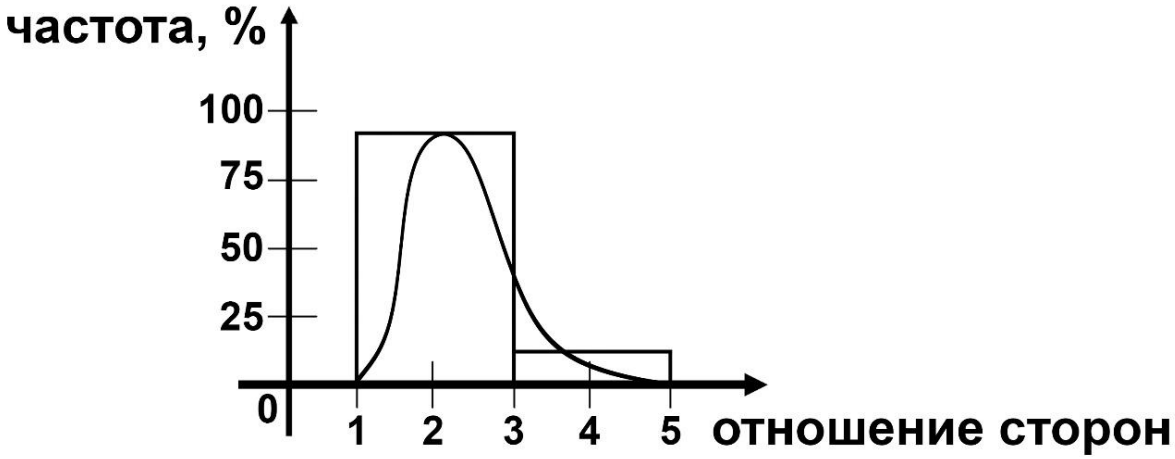

Рисунок 30 – Содержание зерен пластинчатой и игловатой форм составляет 8%, это значит, что сырье относится к I группе лещадности (кубовидной)

На графике видно преобладание кубовидных и остроугольных форм сырья, и маленький процент лещадности, такой материал качественнее, лучше и следовательно его цена на рынке выше.

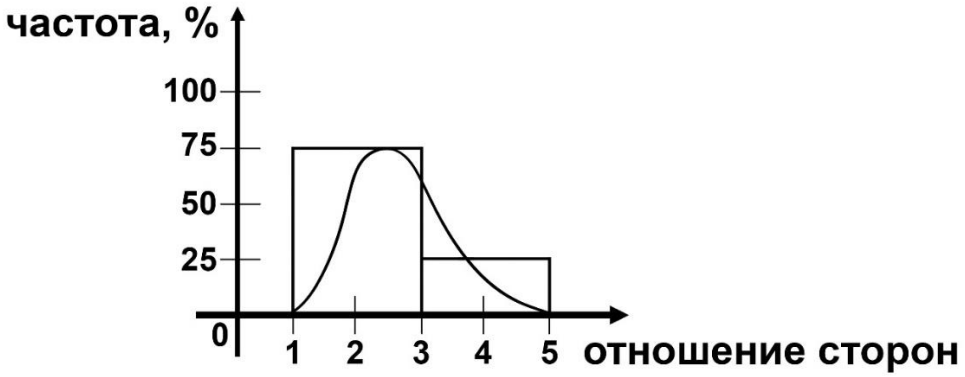

Рисунок 31 – Содержание зерен пластинчатой и игловатой форм составляет 25%, следовательно относится к III или IV группе лещадности (обычная)

В этом же примере видно совсем другое, зерна пластинчатых и игловатых форм составляют немаленький процент относительно «правильных». Это может быть вызвано многими причинами, такими как неправильная транспортировка материала, дробление породы с ошибками в технологии и другими.

Общую картину (совмещение этих графиков) можно увидеть и проанализировать на рисунке 32.

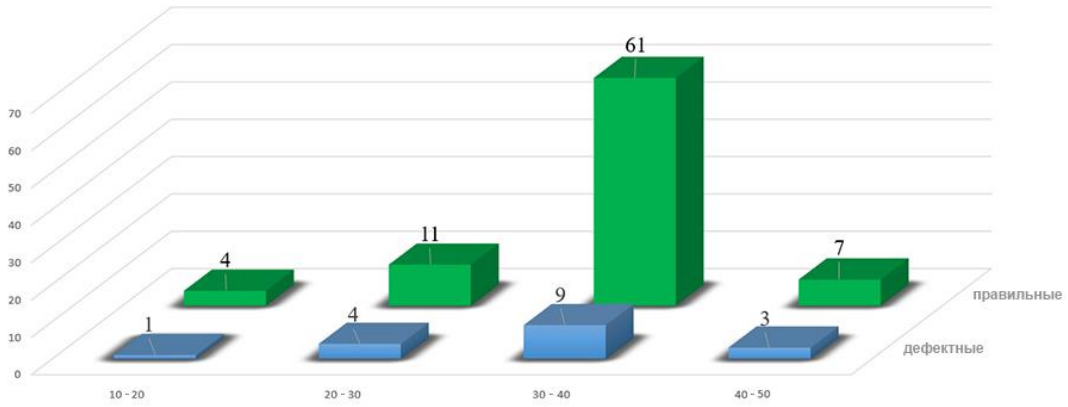

Рисунок 32 – Процентное содержание зерен сырья в зависимости от размера фракций и лещадности

Сам же расчет статистического распределения выборки имеет алгоритм, описанный ниже.

1) Подсчитывается количество интервалов по формуле: k = 1+ 3,2 $lg\ n$, где n – объём выборки. Полученное значение k – округляют до ближайшего целого нечетного числа.

2) Подсчитывается длина каждого интервала:

$h = \Delta x = \frac{x_{max} - x_{min}}{k}$ (с учетом округления).

3) Находится центр гистограммы: $C = \frac{x_{max} + x_{min}}{2}$.

4) Находится начало гистограммы: $a = C - \frac{k}{2} \times h$.

5) Определяются граничные значения каждого i-ого интервала: $[x_i \div x_{i+1} = x_i + h]$, i=1, 2,…k, принимая начало первого интервала $x_1 = a$.

6) Подсчитывается число элементов выборки v_i (v_1; v_2; . . .; v_k), попавших в интервалы h_i, при этом граничные значения x_{i-1} и x_i относят к i.

7) По формуле $f(x_i^*) = \frac{v_i}{n \times h_i}$ (i = 1; 2; . . .; k) вычисляют ординаты оценок плотности вероятности в точках x_i^* – центрах интервалов h_i [11].

2.2 Расчёт общего объёма сырья с заданным фракционным составом на ленточном конвейере

2.2.1 Приближенное описание поверхности полиномами

При хранении габаритных объектов [17] с неровной поверхностью (например, сыпучих) на различных складах возникает задача вычисления объёма. В данной статье предлагается алгоритм расчёта объёма объекта на основе данных, полученных после сканирования поверхности. Результаты сканирования представлены в виде значений узлов регулярной сетки. В представленном алгоритме, который изображен на рисунке 33, предлагается с помощью полиномов Лагранжа аппроксимировать значения высот объекта между узлами сетки с необходимым шагом и вычислять объем путём

сложения прямоугольных параллелепипедов. Прямоугольные параллелепипеды, составляющие объект, имеют высоту, вычисленную с помощью полинома Лагранжа, и площадь основания, равную квадрату шага.

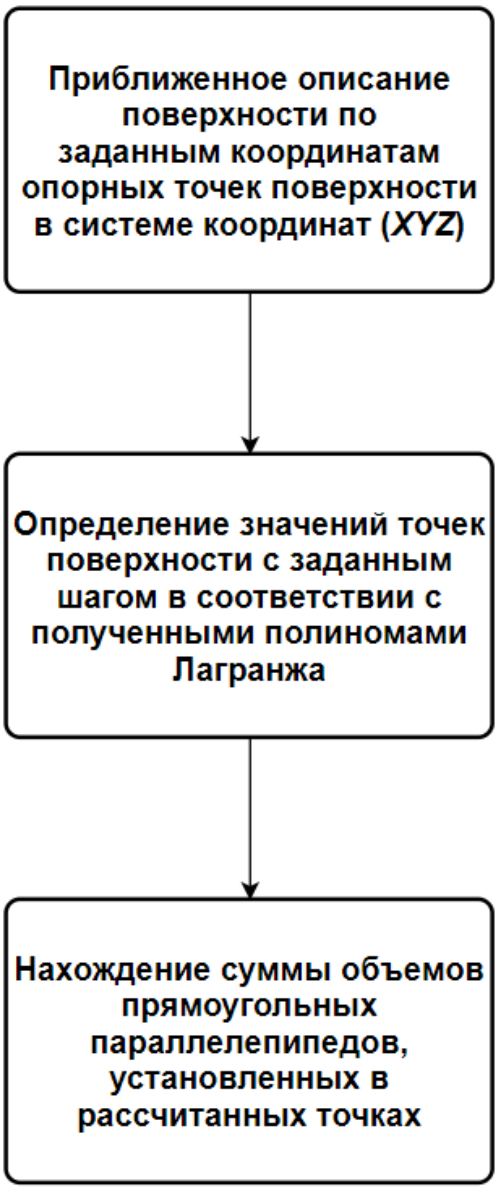

Рисунок 33 – Блок-схема алгоритма вычисление объема технического сыпучего сырья с воздушными пустотами, с помощью полиномов Лагранжа

В результате сканирования объекта мы получаем набор значений высот поверхности, расположенных в узлах регулярной сетки. На рисунке представлен фрагмент объекта размером метр на метр, расстояние между узлами 1 см.

Объем данной фигуры [17] можно представить как сумму объёмов прямоугольных параллелепипедов, которые изображены на рисунке 34. Найти точки, расположенные между узлов сетки, можно с помощью аппроксимации.

Наиболее целесообразно [18] для аппроксимации сложных пространственных поверхностей, заданных координатами узловых точек, применять многомерные полиномы. В отличие от описания поверхности сплайн-функциями, данный метод позволяет исключить колебательный процесс, который возникает в результате совпадения точек поверхности в узловых точках и отсутствия гладкости полученного описания в промежутках между опорными точками поверхности.

В данном случае аппроксимация поверхности между двумя точками основывается на знании координат опорной точки и частных производных в данной точке. Это не накладывает требований на гладкость поверхности между опорными точками [4].

Интерполяционные полиномы Лагранжа одной переменной [17] позволяют аппроксимировать функцию $y=f(x)$ в декартовой системе координат (XY), задаваемую координатами опорных точек (x_i,y_i). Аппроксимированная функция:

$$f(x) = P(x) = \sum_{i=0}^{n} p(x_i) \times \prod_{j=0, j\neq i}^{n} (x - x_j).$$

Где коэффициенты полиномов Лагранжа $p(x_i)$ определяются через значения x_i, y_i в опорных точках. Коэффициент полинома вычисляется по формуле, представленной ниже.

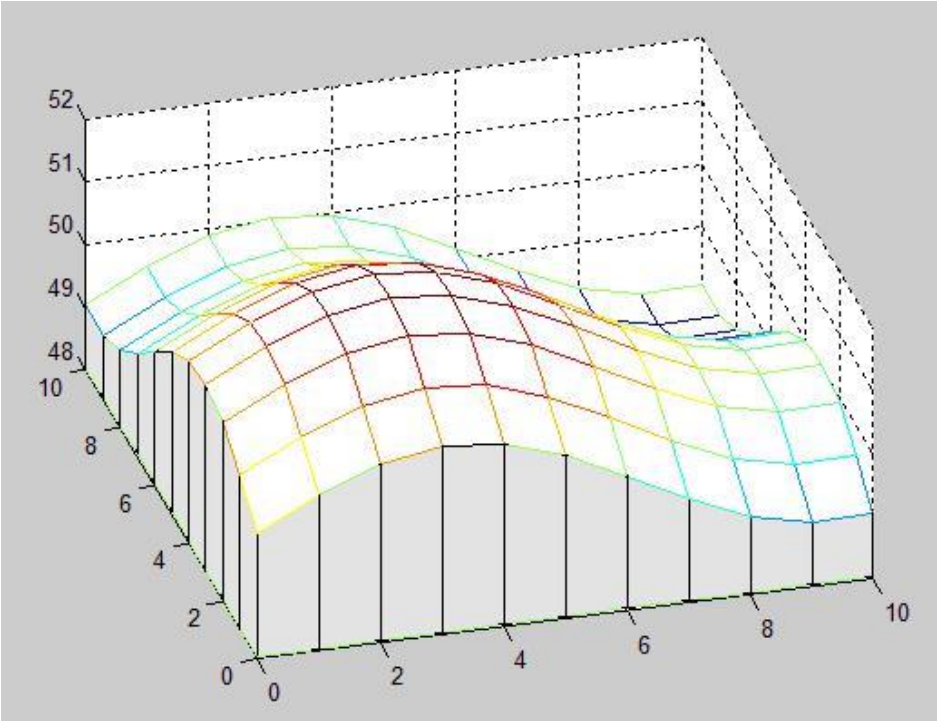

Рисунок 34 – Фрагмент исследуемого объекта

$$p(x_i) = \frac{y_i}{\prod_{j=0, j\neq i}^{n}(x - x_j)},$$

где $i = 0..n$, $j = 0..n$, n – степень полинома.

По аналогии с полиномом одной переменной можно описать полином Лагранжа двух переменных для поверхности, представленной в декартовой системе координат (*XYZ*):

$$z = P(x, y) = \sum_{i=0}^{n}\sum_{j=0}^{m} p(x_i, y_j) \times \prod_{n=0, n\neq i} \prod_{m=0, m\neq j} (x - x_n)(y - y_m),$$

где $i = [0..n]$ – количество опорных сечений поверхности вдоль оси *OX*;

$j = [0..n]$ – количество опорных сечений поверхности вдоль оси *OY*;

$p(x_i, y_i)$ – коэффициенты полинома, определяемые через соответствующие координаты опорных точек поверхности.

$$P(x_i, y_j) = \frac{z_{ij}}{\prod_{n=0,n\neq i}\prod_{m=0,m\neq j}(x - x_n)(y - y_m)}.$$

При интерполяции поверхности полиномами двух переменных необходимо. Чтобы опорные точки поверхности (узлы интерполяции) образовывали сетку. Постоянные коэффициенты полинома $p(x_i, y_i)$ определяются для каждого элемента поверхности через координаты известных опорных точек поверхности и определяются по представленной формуле.

2.2.2 Вычисление объема фрагментов с помощью двойных интегралов

Пусть D – ограниченная замкнутая область в плоскости xOy. Функция $z=f(x,y)$ определена и ограничена в области D. Разобьем эту область на n частей и обозначим их площади соответственно ΔS_1, ΔS_2, ΔS_3, . . . , ΔS_n (рисунок 35). В каждой части ΔS_i возьмем произвольно точку M_i (x_i, y_i) и составим сумму

$$\sum_{i=1}^{n} f(M_i)\Delta S_i.$$

Эта сумма называется интегральной суммой функции $f(x, y)$ в области D.

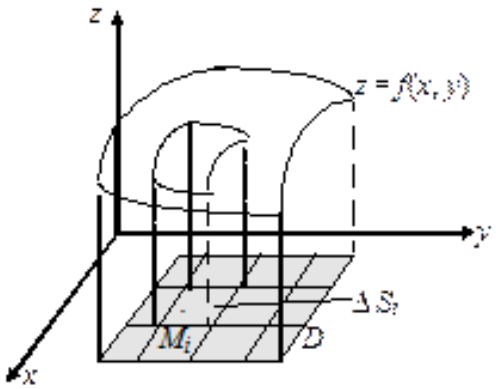

Рисунок 35 – Геометрический смысл двойного интеграла

Определение. Если при любом разбиении области D на части ΔS_i и произвольном выборе точек $M_i(x_i,y_i)$ существует предел последовательности сумм при условии, что максимальный из диаметров частей ΔS_i стремится к нулю, то этот общий предел называют двойным интегралом от функции $f(x, y)$ по области D и обозначают символом

$$\iint_D f(M)dS.$$

Итак,

$$\lim_{max\ \Delta S_i \to 0} \sum_{i=1}^{n} f(M_i)\Delta S_i = \iint_D f(M)dS.$$

Область D называется областью интегрирования, а dS – элементом площади.

Решение этой задачи мы проведем на основе нестрогих геометрических соображений [4]. Однако полученная ниже формула может быть доказана и строгим путем [7].

Разобьем область (Q) на частичные области при помощи прямых, параллельных осям OU и OV. Тогда область (P) разобьется соответствующими координатными линиями на некоторые элементарные криволинейные четырехугольники. Рассмотрим какой-нибудь внутренний элементарный прямоугольник (ΔQ) в плоскости $OU\,V$ с вершинами в точках: где $u > 0$ и $v > 0$ и соответствующий ему элементарный криволинейный четырехугольник (ΔP) в плоскости OXY с вершинами (рисунок 36, 37).

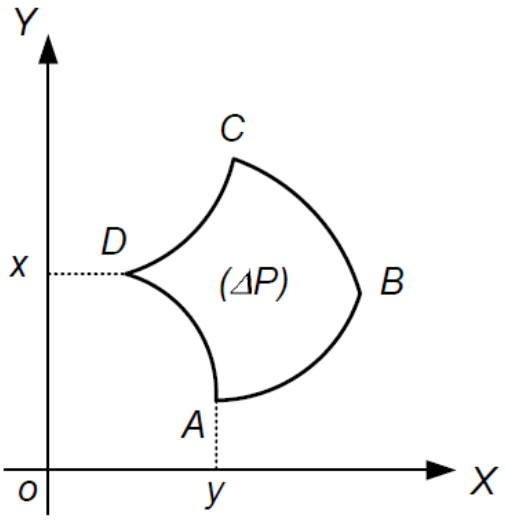

Рисунок 36 – Криволинейный четырехугольник

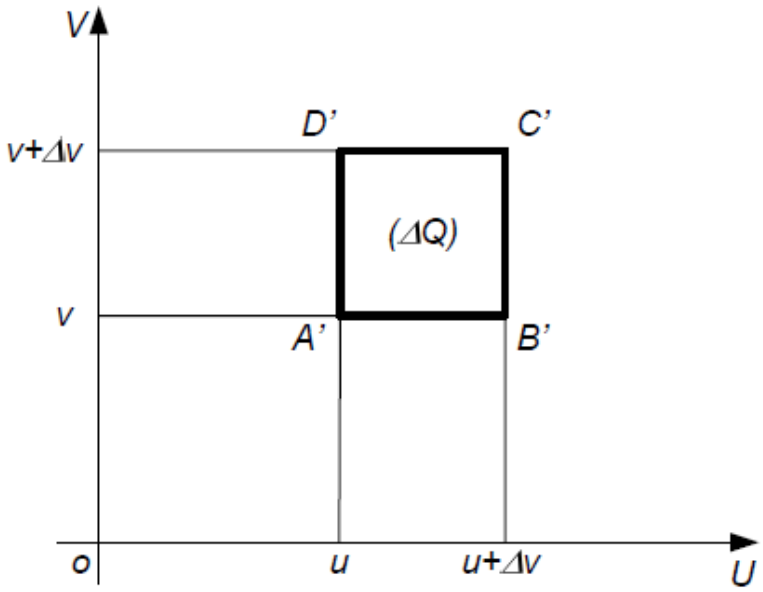

Рисунок 37 – Проекция криволинейного четырехугольника

Найдем его площадь ΔP [6].

Приступая к вычислению площади криволинейного четырехугольника *ABCD*, заметим, что для достаточно малых Δ*v* и Δ*u* дуги его *AB, BC, CD, DA* тоже будут весьма малы, и поэтому приближенно их можно считать прямолинейными; кроме того, приращения функций *x*(*u*, *v*) и *y*(*u*, *v*) с большой точностью можно заменить соответствующими дифференциалами. Таким образом, пренебрегая бесконечно малыми высшего порядка по сравнению с бесконечно малыми *u* и *v*, будем иметь

$$x(u+\Delta u, v) - x(u,v) \approx x'_u(u,v)\Delta u;$$
$$x(u+\Delta u, v) \approx x(u,v) + x'_u(u,v)\Delta u.$$

Аналогично этому будет:

$$x(u+\Delta u, v+\Delta v) \approx x(u,v) + x'_u(u,v)\Delta u + x'_v(u,v)\Delta v.$$

Точно так же:

$$x(u, v+\Delta v) \approx x(u,v) + x'_v(u,v)\Delta v;$$
$$y(u+\Delta u, v) \approx y(u,v) + y'_u(u,v)\Delta u;$$
$$y(u+\Delta u, v+\Delta v) \approx y(u,v) + y'_u(u,v)\Delta u + y'_v(u,v)\Delta v;$$
$$y(u, v+\Delta v) \approx y(u,v) + y'_v(u,v)\Delta v.$$

Тогда вершины четырехугольника *ABCD* [7] можно записать со следующими приближенными значениями координат: $A(x,y), B(x + x'_u\Delta u, y + y'_u\Delta u), C(x + x'_u\Delta u + x'_v\Delta v, y + y_u\Delta u + y'_v\Delta u), D(x + x'_v\Delta v, y + y'_v\Delta v)$, где для краткости положено $x(u, v) = x$, $y(u, v) = y$ и все производные вычислены в точке (u, v). Координаты этих точек показывают, что проекции отрезков *AB* и *CD* на обе оси координат соответственно равны; отсюда следует, что и сами отрезки равны и параллельны. То же можно сказать и об отрезках *AD* и *BC*. Таким образом, с точностью до бесконечно малых высших порядков криволинейный четырехугольник *ABCD* можно рассматривать как обыкновенный параллелограмм. Поэтому площадь четырехугольника (ΔP) приближенно равна удвоенной площади треугольника *ABC*. Из аналитической геометрии известно, что удвоенная площадь треугольника с вершинами $A(x_1y_1)$, $B(x_2, y_2)$, $C(x_3, y_3)$ равна абсолютной величине определителя

$$\begin{vmatrix} x_2 - x_1 & y_2 - y_1 \\ x_3 - x_2 & y_3 - y_2 \end{vmatrix}.$$

В данном случае, согласно этой формуле, площадь криволинейного четырехугольника *ABCD* (с точностью до бесконечно малых высшего порядка) равна абсолютной величине определителя

$$\begin{vmatrix} x'_u \Delta u & y'_u \Delta u \\ x'_v \Delta v & y'_v \Delta v \end{vmatrix} = \begin{vmatrix} x'_u & y'_u \\ x'_v & y'_v \end{vmatrix} \Delta u \Delta v.$$

Введем обозначение $I = (u, v) = \begin{vmatrix} x'_u & y'_u \\ x'_v & y'_v \end{vmatrix}$

Такой определитель называется якобианом. Иначе его называют функциональным определителем функции $x(u, v)$, $y(u, v)$.

Таким образом, из вышесказанного следует, что значение площади *AP* элементарного четырехугольника выразится приближенным равенством

$$\Delta P \approx |I(u, v)| \Delta u \Delta v.$$

Выражение в правой части обычно называется элементом площади в криволинейных координатах.

Например [13], в случае преобразования полярных координат имеем:

$$x'_r = cos\theta, x'_\theta = -r\ sin\theta, y'_\theta = r\ cos\theta;$$
$$I(r, \theta) = \begin{vmatrix} cos\theta & sin\theta \\ -r\ sin\theta & r\ cos\theta \end{vmatrix} = r.$$

Значит, для элемента площади в полярных координатах получим выражение $rdrd\Theta$, с которым мы уже встречались ранее.

Так как $\Delta u\ \Delta v = \Delta Q$ — площадь прямоугольника *A'B'C'D'*, то находим

$$|I(u, v)| \approx \frac{\Delta P}{\Delta Q}.$$

Это приближенное равенство будет тем точнее, чем меньшее Δ и $u\Delta v$. Следовательно, в пределе при стремлении Δu и Δv к нулю, то есть при сжатии области (ΔQ) в точку (u, v) получим точное равенство:

$$|I(u,v)| = \lim \frac{\Delta P}{\Delta Q}.$$

Величина $|I\ (u,\ v)|$ показывает [14], во сколько раз увеличивается или уменьшается элемент площади в окрестности точки (u, v) плоскости $OU\ V$ при отображении ее в окрестность соответствующей точки (x, y) в плоскости OXY; таким образом, абсолютная величина функционального определителя (якобиана) является как бы коэффициентом ”растяжения” области (Q) (в данной ее точке (u, v)) при отображении ее на область (P).

Просуммируем площади всех элементарных четырехугольников, получим приближенное выражение для площади фигуры (P):

$$P \approx \sum |I(u,v)|\Delta u \Delta v.$$

Это равенство тем точнее, чем мельче разбиение области (Q) (а вместе с ней и области (P)). Переходя к пределу при $max\ \Delta u \to 0$ и $max\ \Delta v \to 0$, мы получим точное равенство. Но сумма в правой части формулы представляет интегральную сумму для интеграла

$$\iint\limits_{(Q)} |I(u,v)|\, dudv,$$

из которой лишь выброшены слагаемые, отвечающие «неправильным» участкам, расположенным вдоль контура области (Q). Поскольку доля, вносимая в интегральную сумму «неправильными» участками, становится сколь угодно малой по мере того, как разбиение становится все более и более

мелким, переход к пределу в приближенном равенстве приводит нас к точной формуле

$$P = \iint\limits_{(Q)} |I(u,v)|\, dudv.$$

По итогу, если область интегрирования R представляет собой прямоугольник *[a,b]×[c,d]*, изображенный на рисунке 38.

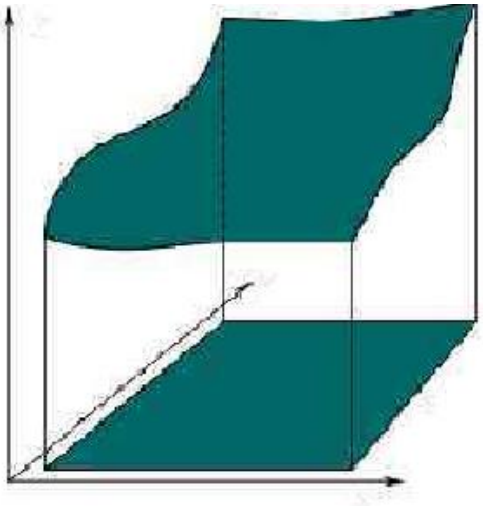

Рисунок 38 – Представление одной части сетки

Тогда двойной интеграл в такой области выражается через повторный интеграл в следующем виде:

$$\iint\limits_R f(x,y)dxdy = \int\limits_a^b \left(\int\limits_c^d f(x,y)dy \right) dx = \int\limits_c^d \left(\int\limits_a^b f(x,y)dx \right) dy.$$

В данном случае область интегрирования R относится одновременно к типу I и II, так что у нас есть возможность выбирать, по какой переменной (*x* или *y*) начинать интегрировать функцию *f(x,y)*. Обычно удобнее начинать с более простого интеграла.

В частном случае, когда подынтегральная функция *f(x,y)* "расщепляется" на произведение *g(x)h(y)*, двойной интеграл равен произведению двух определенных интегралов:

$$\iint\limits_R f(x,y)dxdy = \iint\limits_R g(x)h(y)dxdy = \left(\int\limits_a^b g(x)dx\right)\left(\int\limits_c^d h(y)dy\right).$$

После рассчитанного объема, который вычислялся с пустотами технического сырья, следует применить насыпную плотность каждого вида зерен, которая была получена в предыдущем разделе, так как благодаря ей будет известен истинный объем (рисунок 39).

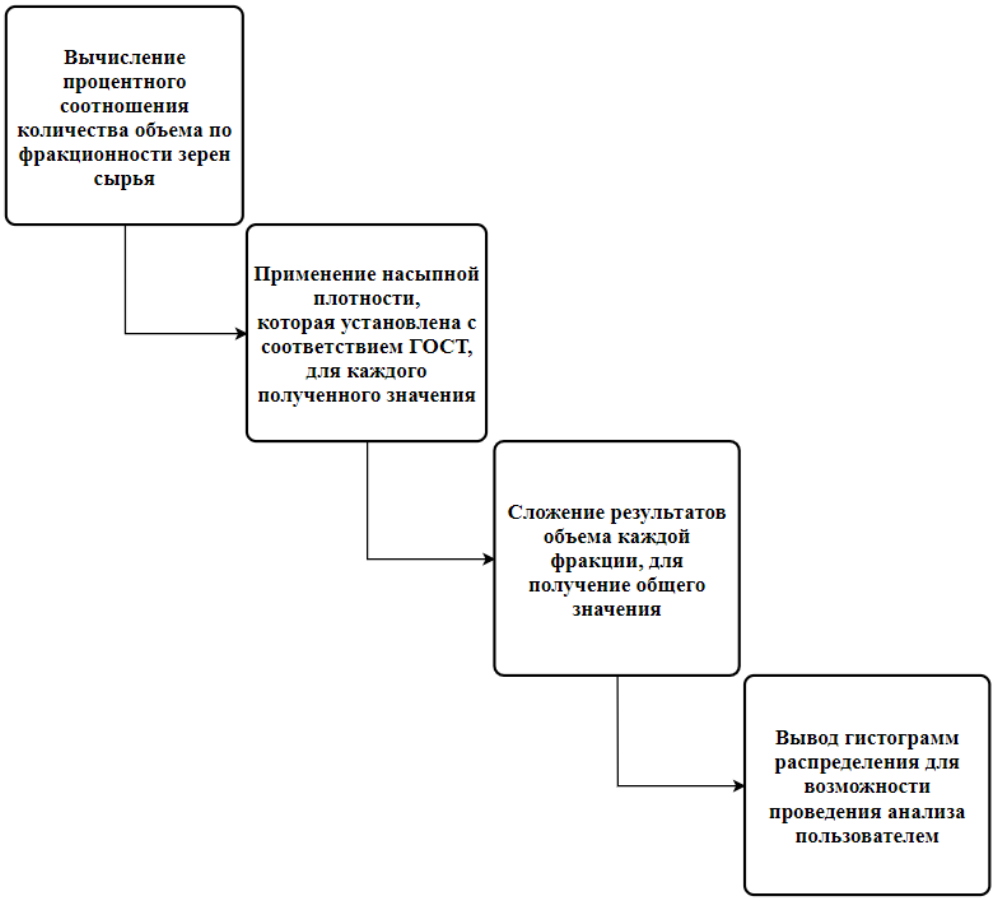

Рисунок 39 – Блок-схема получения истинного объема с помощью полученных распределений характеристик материала

Зная процентное содержание фракционного состава сырья, будет не сложной задачей вычислить количество объема для каждой фракции. Затем полученное значение применяется для насыпной плотности каждого, что после, при сложение полученных данных и даст истинный объем технического сырья [6].

3 Программная реализация расчета объема технического сырья на платформе asp.net.core

Рассмотрим алгоритм работы программы, вычисляющей объем фигуры, заданной дискретными значениями [21].

Исходные данные (результат сканирования) представлены двумерным массивом.

Номер строчки соответствует координате *x* в декартовой системе координат, а номер столбца – координате *y*. Значения *x* и *y* изменяются с шагом 1, что соответствует расстоянию между точками съёмки. Таким образом, элементы представленной матрицы соответствуют высоте заданной точки или координате *z* в декартовой системе координат (*XYZ*).

Вычисление элементов массива $p(x_i; y_i)$ реализуется в соответствии с формулой описанной в пункте №2.2.1. Алгоритм вычисления представлен блок-схемой (рисунок 40).

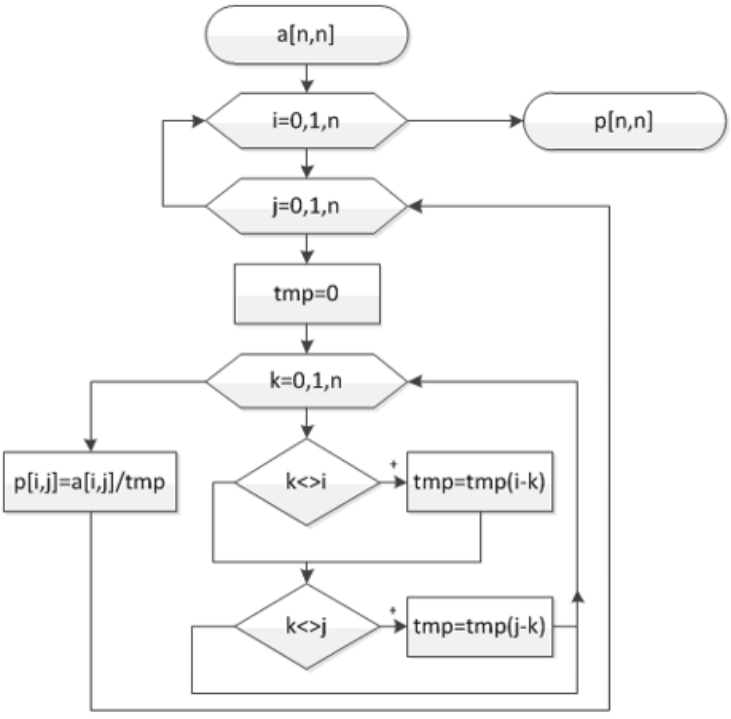

Рисунок 40 - Блок-схема подпрограммы расчёта коэффициентов полинома Лагранжа

На входе в подпрограмму подаётся массив исходных данных $a[n; n]$. В цикле с параметром k определяется знаменатель для очередного коэффициента полинома. В результате коэффициент определяется как отношение соответствующего элемента исходного массива к вычисленному знаменателю *tmp*.

Теперь, когда коэффициенты Лагранжа получены, можно вычислить объем фигуры путём сложения прямоугольных параллелепипедов с квадратным основанием заданного размера. На рисунке 41 представлена блок-схема подпрограммы расчёта объёма объекта с заданным шагом h, который определяет сторону квадратного основания прямоугольного параллелепипеда.

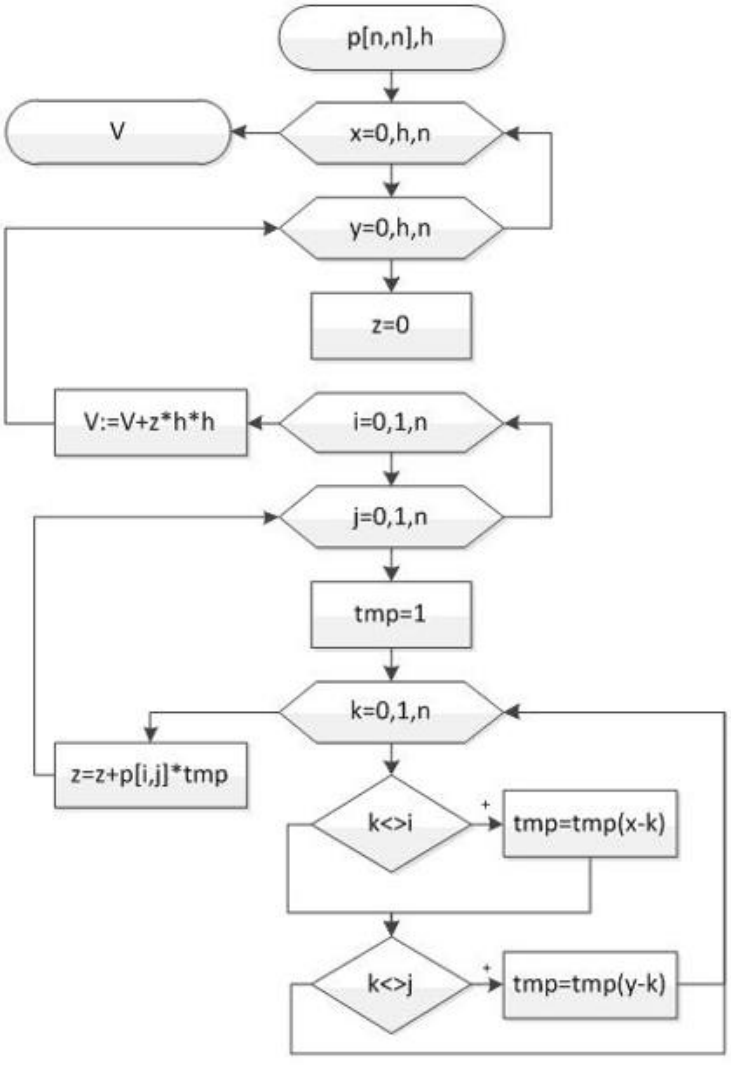

Рисунок 41 – Блок-схема подпрограммы расчёта объёма объекта с заданным шагом

Вычисление сводится к двум этапам [15]:

- определение высоты прямоугольного параллелепипеда;
- добавление объёма очередного параллелепипеда в накопитель.

На входе в подпрограмму подаются размер шага h и матрица коэффициентов полинома Лагранжа $p[n; n]$. Начиная с точки (0; 0; z_{00}), определяется величина z_{ij}, где $i = [0::n]$ – значение координаты оси OX, а $j = [0; n]$ – значение координаты по оси OY.

Объём прямоугольного параллелограмма вычисляется традиционно путём умножения высоты на ширину и длину ($h \times h \times z_{ij}$).

После определения объёма очередного прямоугольного параллелограмма, полученная величина добавляется в накопитель V. В результате перебора всех точек (x; y) на заданном поле с шагом h в переменной V получаем искомый объём.

Задача вычисления интеграла $S = \int_a^b f(x)dx$ также относится к классическим задачам математики и программирования. Вычисление интеграла аналогично задаче суммирования, поскольку практически вычисление значения S сводится к вычислению суммы $S = \sum_k f(x_k)delta_x$.

Решение библиотеки *csharp_int* содержит проекты, позволяющие провести сравнительный анализ эффективности последовательных и параллельных алгоритмов вычисления интеграла.

Структура решения [14].

Решение *csharp_int* содержит 2 проекта. Следуя принципу разделения интерфейса и бизнес-логики, содержательная часть реализована в проекте *ClassLibraryIntegral*, представляющем динамическую библиотеку классов. Библиотека содержит один класс – *Integral*, включающий методы, реализующие последовательные и параллельные алгоритмы вычисления интеграла. Параллельный алгоритм построен на использовании инструментария *Parallel.For*.

Для распараллеливания используется стратегия сегментного алгоритма вычисления суммы. Интервал интегрирования разбивается на k сегментов, на

каждом из которых для вычисления интеграла используется обычный последовательный алгоритм. Вычисление интегралов на отдельных сегментах ведется параллельно. Эта стратегия оказывается весьма эффективной, даже в том случае, когда вычисление интеграла на каждом сегменте ведется последовательно. Выигрыш по времени может достигать десятки раз при относительно небольшой потери точности. В учебном курсе поясняется причина такой эффективности. Подынтегральная функция на каждом участке ведет себя более гладко, в сравнении со всем интервалом интегрирования. По этой причине на каждом сегменте достаточно быстро достигается нужная точность вычислений.

Что касается последовательного метода вычисления интеграла на отрезке, то в классе *Integral* представлены два классических метода – более простой в реализации метод прямоугольников и более эффективный метод трапеций.

Для расчетов статистических данных фракционности и группы лещадности материала задействована библиотека *NAG* для *.NET*, которая включает в себя математические и статистические методы платформы *Microsoft .NET*.

Что касается статистики, то библиотека поддерживает множество разных видов статистических распределений, умеет строить линейную и нелинейную регрессии, поддерживает различные тесты для определения правильности статистических гипотез, проводит кластерный анализ.

Одной из самых важных функций этой библиотеки для анализа размеров и форм сырья является *g01ae*, так как он строит частотное распределение переменной в соответствии с предоставленными пользователем или вычисленными методом граничными значениями класса.

```
public static void g01ae(
 int n,
 int k,
 double[] x,
 int iclass,
 double[] cb,
```

```
 int[] ifreq,
 out double xmin,
 out double xmax,
 out int ifail
)
```

Параметры:

n – количество наблюдений;

k – число классов, требуемых в частотном распределении. Независимо от того, заданы ли пользователем граничные значения класса, к должны включать в себя два экстремальных класса;

X – выборка наблюдений переменной, для которой требуется частотное распределение;

iclass – указывает, следует ли вычислять значения границ класса в пределах *g01ae*, которые предоставлены.

cb если *iclass*=0, то элементы из *cb* не нужно назначать значения, так как *g01ae* вычисляет граничные значения класса. Если *iclass*=1, то первый элемент *cb* должен содержать указанные граничные значения классов в порядке возрастания;

ifreq – элементы системы *ifreq* содержать частоты в каждом классе;

xmin – наименьшее значение в выборке;

xmax – самое большое значение в выборке;

ifail – *ifail* = 0, если только метод не обнаруживает ошибку.

С помощью полученных расчетов, благодаря функции и её реализации, проводится визуализация *JavaScript* с использованием библиотеки *D3.js*.

D3.js представляет библиотеку на языке *JavaScript* для обработки и визуализации данных. Само название *D3* расшифровывается как *Data-Driven Documents* и как бы делает упор на управление данными, хотя ключевой функциональностью библиотеки являются мощные возможности для их визуализации.

Библиотека *D3.js* основана прежде всего на использовании *JavaScript*, *SVG* и *CSS* в противовес другим подобным библиотекам, которые вместо *SVG* используют элемент *canvas* и его возможности. Если стандартные механизмы

рисования, например, элемент *canvas*, полагаются на пиксели, то svg использует векторы. Применение *SVG* позволяет создавать структуры с насыщенной графикой, обладающие анимацией и возможностями взаимодействия.

Ключевым аспектом библиотеки *D3.js* является работа с данными. Для добавления данных в элемент применяется метод *data*(). В качестве аргумента в этот метод передается массив объектов.

Ключевой задачей при визуализации данных является их сопоставление с визуальными элементами. Чтобы упростить процесс сопоставления *D*3 предоставляет специальные конструкции, которые называются *scale*. Однако роль *scale* только сопоставлением не ограничивается, они также могут служить в качестве строительных кирпичиков более сложных конструкций в *d*3.

Scale можно представить в качестве математической функции, которая преобразует некоторое значение из одного интервала, который называется *domain*, в значение, принадлежащее другому интервалу, который называется *range*.

Применение *SVG* позволяет легко рисовать простейшие графические примитивы и затем из них складывать более сложные фигуры. Отличительной особенностью *SVG* является то, что эта технология позволяет применять стили *CSS* для настройки визуализации фигур, что дает нам дополнительный контроль над визуализацией.

Гистограммы (*bar chart*) отображают данные в виде столбиков. В *D*3 каждый отдельный элемент диаграммы можно выразить через объект *rect*.

Важной составляющей программной реализации является визуализация. Для того, чтобы была возможность что-либо отображать с *three.js*, нужны три вещи: сцена (*scene*), камера (*camera*) и визуализатор (*renderer*) – также называемый рендерер, чтобы была возможность показывать сцену, снятую камерой.

```
var scene = new THREE.Scene();
```

```
    var camera = new THREE.PerspectiveCamera( 75,
window.innerWidth / window.innerHeight, 0.1, 1000 );
    var renderer = new THREE.WebGLRenderer();
    renderer.setSize( window.innerWidth,
window.innerHeight );
    document.body.appendChild( renderer.domElement );
```

Камера с перспективной проекцией, область просмотра параметры которой изображены на рисунке 42.

Этот режим проекции предназначен чтобы наиболее полно сымитировать человеческое зрение. Это самый распространенный режим проецирования, используемый для визуализации (рендеринга) трехмерной (3*D*) сцены.

Конструктор:

`PerspectiveCamera(fov, aspect, near, far).`

`fov` – вертикальный угол области просмотра камеры;

`aspect` – соотношение сторон области видимости камеры;

`near` – ближняя плоскость отсечения области видимости камеры;

`far` – дальняя плоскость отсечения области видимости камеры.

Вместе они определяют область просмотра камеры (в виде усеченной пирамиды)

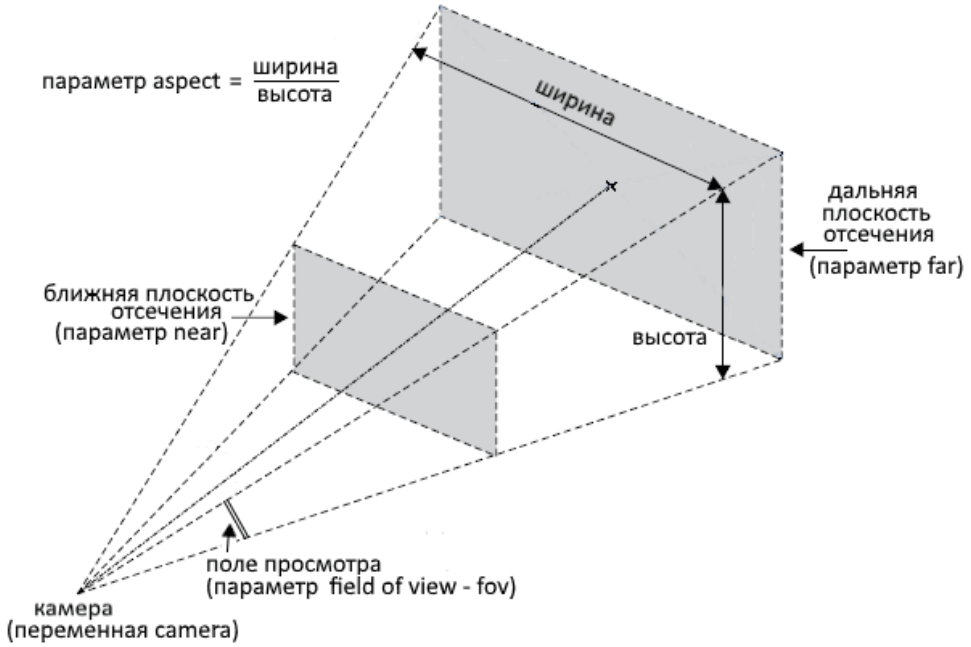

Рисунок 42 – Область просмотра перспективной камеры

Ортографическая камера, область просмотра параметры которой изображены на рисунке 43.

Камера с ортографической проекцией. При этом способе проецирования, размер объекта в отображаемой картинке остается постоянным, независимо от расстояния между ним и камерой.

Конструктор:

`OrthographicCamera( left, right, top, bottom, near, far ).`

`left` – левая плоскость отсечения области видимости камеры;

`right` – правая плоскость отсечения области видимости камеры;

`top` – верхняя плоскость отсечения области видимости камеры;

`bottom` – нижняя плоскость отсечения области видимости камеры;

`near` – ближняя плоскость отсечения области видимости камеры;

`far` – дальняя плоскость отсечения области видимости камеры.

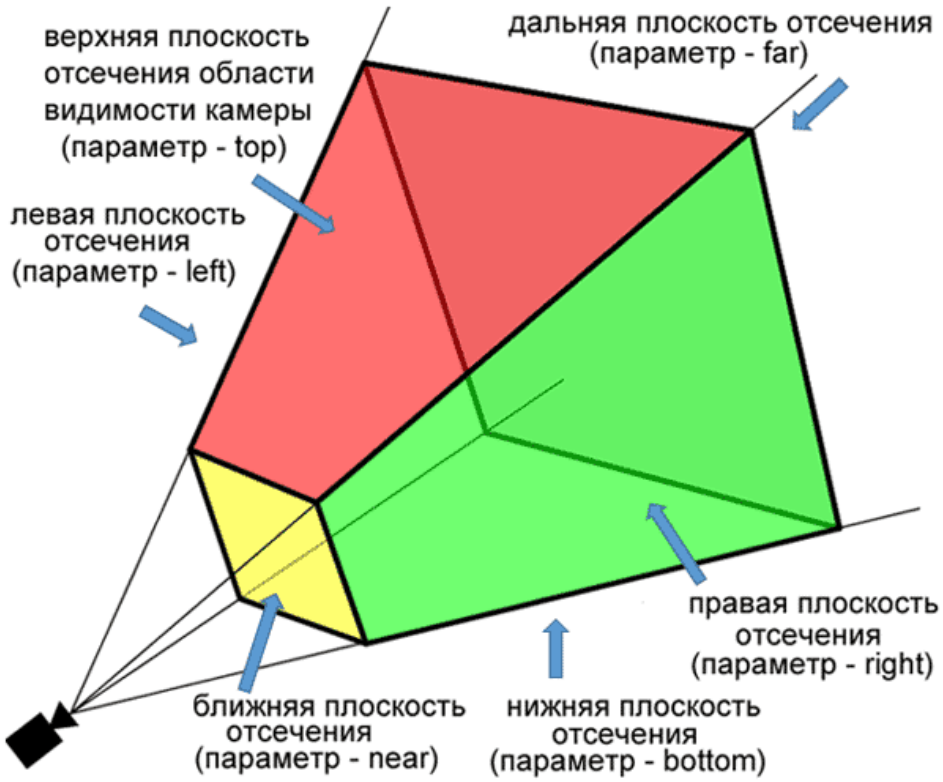

Рисунок 43 – Область просмотра ортографической камеры

Управление орбитами.

Орбитальный контроль (элемент управления) позволяет камере вращаться вокруг цели. Для его использования, как и всех файлов в директории */examples*, нужно будет отдельно включить этот файл в *HTML*-код.

Конструктор:

```
OrbitControls( object-объект') "onmouseout=" Hide
()" > Object>, domelement )
```

`object - объект')" onmouseout="hide()">object` – камера, которой нужно управлять (обязательный аргумент).

`domElement` – элемент HTML, используемый для прослушивателей (приемников) событий (дополнительный, необязательный аргумент). По умолчанию это весь документ целиком, однако, если нужно чтобы элементы управления работали от какого-либо определенного элемента (например, <*canvas*> – холст).

Перейдем к *WebGLRenderer*, который отображает обработанные сцены с помощью *WebGL*.

Конструктор:

```
WebGLRenderer( parameters ).
```

Parameters – объект со свойствами, определяющими поведение визуализатора (рендерера) (дополнительный, необязательный параметр).

Конструктор также допускает полное отсутствие параметров. Во всех подобных случаях, им будут приниматься соответствующие значения по умолчанию.

Допустимы следующие параметры:

- *canvas* – холст, на который визуализатор выводит результаты своей работы. Он соответствует свойству *domElement*, указанному ниже. Если он не был передан здесь, то будет создан новый элемент *canvas*;

- *context* – этот параметр можно использовать для присоединения визуализатора (рендерера) к существующему *RenderingContext*. Значением по умолчанию является *null*;

- *precision* – точность шейдера. Может быть *highp* (сокращение от английских слов *high precision* – высокая точность), mediump (от слов *medium precision* – средняя точность) или *lowp* (от слов *low precision* – низкая точность). Значением по умолчанию является *highp*, если поддерживается устройством;

- *alpha* – параметр определяет, будет ли холст (*canvas*) содержать буфер альфа-канала (канал прозрачности) или нет. Значением по умолчанию является *false*;

- *antialias* – параметр определяет, будет ли выполняться сглаживание (*antialiasing*). Значением по умолчанию является *false*;

- *stencil* – будет ли в буфере рисования находиться буфер трафарета не меньше 8 бит. Значением по умолчанию является *true*.

На рисунке 44 видно, что благодаря правильному использованию вышеперечисленного получаем скриншоты выполнения разработанной программы, , где мы видим параметр скорости движения конвейера и частоту кадров съемки лазерного измерительного комплекса, также вывод искомого истинного объема сыпучего сырья [5], которое изображено в сечении и 3*D* модели, что позволит следить за корректной работой прибора.

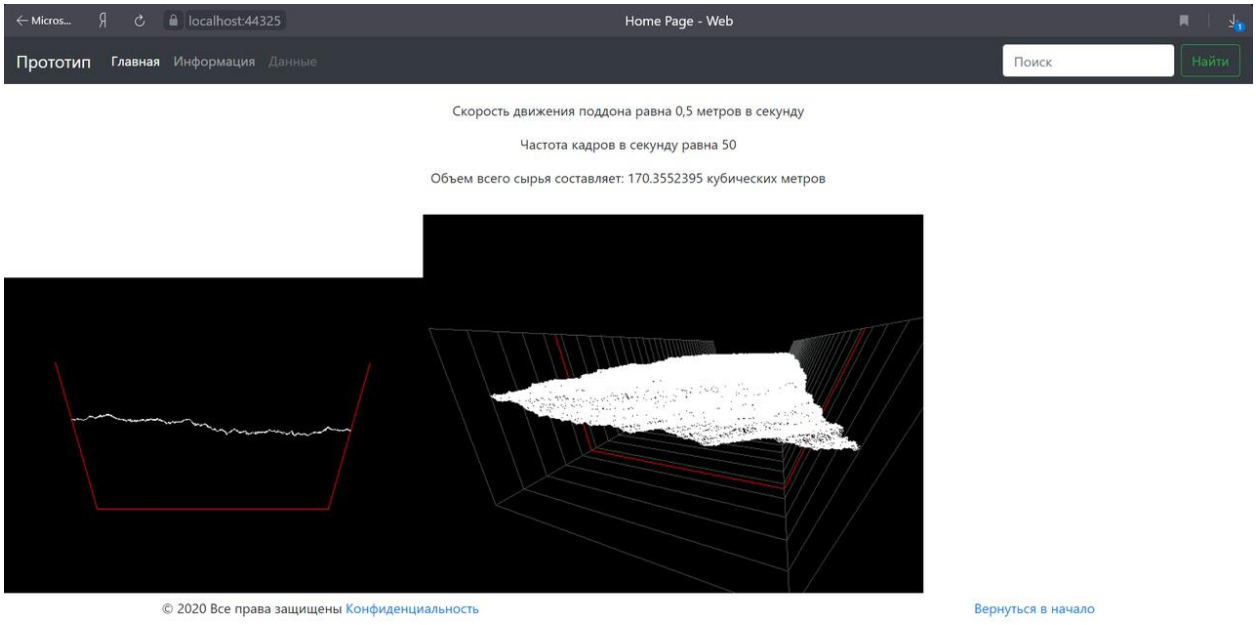

Рисунок 44 – Вывод веб-приложения в браузер. Сечение и 3*D* модель

Зайдя во вкладку, «информация», как показано на рисунке 40, можно увидеть многомерную гистограмму, построенную проведенных измерений лещадности и фракцинного состава.

Рисунок 45 – Вывод гистограммы, где правильные – это кубические и остроугольные, дефектные – пластинчатые и игольчатые, отношение сторон которых больше 3

С помощью такого распределения, есть возможность более точно скорректировать дальнейшие действия на предприятии, провести анализ качества поставляемого сырья и оценить еще множество характеристик и факторов.

ЗАКЛЮЧЕНИЕ

В ходе магистерской диссертации был изучен теоретический материал по таким темам, как: алгоритмы расчета объема технического сырья с учетом его основных факторов, приборы и их устройство как программное так и математическое, веб-программирование с использованием библиотек.

Также разработан алгоритм расчета объема технического сырья с помощью полиномов Лагранжа и двойных интегралов, с проведенным анализом фракционного состава материала.

Реализовано программное обеспечение с помощью объектно-ориентированного программирования на платформе *asp.net core*, что позволяет сотруднику организации имея выход к сети интернет с любого удобного для него устройства, измерять объем технического сырья, просматривать его фракционность на гистограммах, и все это в режиме реального времени.

По полученным в результате проведённого эксперимента данным можно сделать вывод о том, что точность вычислений тем выше, чем меньше шаг и, следовательно, чем больше точек задействовано в вычислениях. Но большое количество точек увеличивает сложность алгоритма, что заметно при больших массивах данных.

В целом, с помощью проведенной работы и разработанного алгоритма можно рассчитать объём технического сырья. В дальнейшем планируется предпринять действия по уменьшению погрешности вычислений, увеличению быстродействия алгоритма, а также его усовершенствования, применяя разные методы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Горчаков Г.И.Строительные материалы /Г. И. Горчаков, Ю. М. Баженов. – М.: Стройиздат, 1986
2. Домокеев А.Г. Строительные материалы / А. Г. Домокеев. –М.: Высшая школа, 1989
3. Мордасов, Д.М . Технические измерения плотности сыпучих материалов: Учеб.пособие. / Мордасов М.М . // Тамбов: Изд-во Тамб, гос. техн. ун-та, 2004
4. Колдаев В.Д. Численные методы и программирование: учебное пособие. М.: Форум. ИНФРА-М, 2008
5. Пискунов, Н. С. Дифференциальное и интегральное исчисления: в 2 т. / Н. С. Пискунов. – Т. 2. – М.: Наука, 1985
6. Шестаков, А. А. Курс высшей математики / А. А. Шестаков. – М.: Высш. шк., 1981
7. Аминов Ю.А. Дифференциальная геометрия и топология кривых. – М.:Наука, 1987
8. Бурмистров, К.В. Совершенствование процесса формировании и эксплуатации усреднительных складов на горнодобывающих предприятиях путем применения 2D-сканирования / Цуприк Л.С., Бурмистрова И.С. // Комбинированная агротехнология: устойчивое и экологически сбалансированное освоение недр: материалы международной научно-технической конференции, г. Магнитогорск, 2015: - Сб. тезисов. - Магнитогорск: МГТУ, 2015. -С. 45-46
9. Жуков В.К. Теория погрешностей технических измерений: уч. пособие/ ТПУ. – Томск: Изд-во ТПУ, 2009. 180 с
10. Гмурман, В. Е. Теория вероятностей и математическая статистика: [учебное пособие для студентов вузов] / В. Е. Гмурман. 7-е изд., стер. - М. : Высшая школа, 1999. - 478с
11. Андерсон Т. В. Введение в многомерный статистический анализ. М., 1963

12. Кендалл М. Дж., Стьюарт Ф. Многомерный статистический анализ и временные ряды. М., 1976; Прикладная статистика: классификация и снижение размерности. М., 1989

13. Аверченков В.И. Основы математического моделирования технических систем/ В.И. Аверченков, В.П. Федоров, М.Л. Хейфец — Москва: Флинта, 2011 г.— 271 с. — Электронное издание. Режим доступа: http://ibooks.ru, свободный. – Загл. с экрана. (Дата обращения: 15.01.2020)

14. В. М Вержбицкий. Численные методы. Линейная алгебра и нелинейные уравнения. М., Высшая школа 2000

15. ГОСТ 8.401-80 Классы точности средств измерений. Общие требования

16. ГОСТ 8.207-76 Государственная система обеспечения единства измерений. Прямые измерения с многократными наблюдениями. Методы обработки наблюдений. Общие положения

17. ГОСТ 8269.0-97 Щебень и гравий из плотных горных пород и отходов промышленного производства для строительных работ

18. Руководство asp.net core [Электронный ресурс]. – 2020. – Режим доступа: https://metanit.com/sharp/aspnet5/1.1.php, свободный. – Загл. с экрана. (Дата обращения: 05.03.2020)

19. Высокоточное измерение объёмов сыпучих материалов и инвентаризация складов сырья [Электронный ресурс]. – 2020. – Режим доступа: http://www.ngce.ru/izmerenie_obemov_sypuchih_materialov.html, свободный. – Загл. с экрана. (Дата обращения: 21.01.2020)

20. Материаловедение. Строительные материалы : лабораторный практикум / В. К. Широкородюк [и др.]. – Краснодар : КубГАУ, 201

21. Широкородюк В. К. Строительные материалы / В. К. Широкородюк. – Краснодар : КубГАУ, 2013

22. Тахеометр: Угловые величины, Горизонтальное направление, Вертикальный угол [Электронный ресурс]. – 2020. – Режим доступа:https://greleon.ru/geodpribory/litrapribory/324-taheometr-leica-tps400-

uglovye-velichiny-gorizontalnoe-napravlenie-vertikalnyy-ugol.html, свободный. – Загл. с экрана. (Дата обращения: 03.04.2020)
23. three.js Справочное руководство [Электронный ресурс]. – 2020. – Режим доступа: https://documentation.help/three.js-ru/overview.html, свободный. – Загл. с экрана. (Дата обращения: 15.11.2019)

ПРИЛОЖЕНИЕ А

Руководство программиста

HomeController

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Web.Models;

namespace Web.Controllers
{
    public class HomeController : Controller
    {
        public IActionResult Index()
        {
            Point2 y = new Point2();
            return View(y.Massiv());
        }

        public IActionResult Privacy()
        {
            return View();
        }

        [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]
        public IActionResult Error()
        {
            return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });
```

```
        }

    }
}
```

ErrorViewModel

```
using System;

namespace Web.Models
{
    public class ErrorViewModel
    {
        public string RequestId { get; set; }

        public bool ShowRequestId => !string.IsNullOrEmpty(RequestId);
    }
}
```

POINT

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Drawing;

namespace Web.Models
{
    public class Point2
    {
        public int[,] Massiv(int n = 600)
```

```
        {
            int[,] Tochki = new int[2000,600];
            Random ran = new Random();
            for (int j = 0; j < 2000; j++)
            {
                if (j == 0)
                {
                    for (int i = 0; i < 600; i++)
                    {
                        if (i == 0)
                        {
                            Tochki[j, i] = ran.Next(-110, 130);
                        }
                        else
                        {
                            if  ((i>300)  &&  (Tochki[j,i-1]+i  -
440)>300)
                            {
                                Tochki[j, i] = Tochki[j, i - 1] -
1;
                            }
                            else
                            {
                                Tochki[j, i] = Tochki[j, i - 1] +
ran.Next(-1, 2);

                            }
                        }
                    }
                }
                else
                {
                    for (int i = 0; i < 600; i++)
                    {
                        if (i == 0)
```

```
                        {
                            Tochki[j,  i]  =  Tochki[j-1,  i]  +
ran.Next(-1, 2);
                        }
                        else
                        {
                            if ((i > 300) && (Tochki[j, i - 1] +
i - 440) > 300)
                            {
                                Tochki[j, i] = Tochki[j, i - 1] -
1;
                            }
                            else
                            {
                                int min = Tochki[j, i - 1];
                                int max = Tochki[j - 1, i];
                                if (min == max)
                                {
                                    Tochki[j,   i]   =   min   +
ran.Next(-1, 2);
                                }
                                else if (min > max)
                                {
                                    min = Tochki[j - 1, i];
                                    max = Tochki[j, i - 1];
                                    Tochki[j, i] = ran.Next(min,
max + 1);
                                }
                                else
                                {
                                    Tochki[j, i] = ran.Next(min,
max + 1);
                                }
                            }
                        }
                    }
```

```
                }
            }
            return Tochki;
        }
    }
}
```

STAT

```
using System;
using NagLibrary;
using System.Globalization;
using System.IO;
namespace NagDotNet
{
      public class G01AEE
      {
      static string datafile = "Data/g01aee.d";
      static void Main(String[] args)
      {
            if (args.Length == 1)
            {
                  datafile = args[0];
            }
            Start ();
      }
      public static void Start ()
      {
            try
            {
                  DataReader sr = new DataReader(datafile);
                  double xmax, xmin; int i, iclass, j, k, n,
nprob;
                  int ifail;
                  Console.WriteLine("g01ae Program Results");
                  sr.Reset();
                  sr.Reset();
                  nprob = int.Parse(sr.Next());
                  for (i = 1; i <= nprob; i++)
                  {
                        sr.Reset();
                        n = int.Parse(sr.Next());
                        iclass = int.Parse(sr.Next());
                        k = int.Parse(sr.Next());
                        double[] cb = new double[k+2];
                        double[] x = new double[n];
                        int[] ifreq = new int[k+2];
```

```
                        if ( n >= 1 && k >= 2)
                        {
                              sr.Reset();
                              for (j = 1; j <= n; j++)
                              {
                                    x[j - 1] =
double.Parse(sr.Next(), CultureInfo.InvariantCulture);
                              }
                              Console.WriteLine("");
                              Console.WriteLine(" {0}{1,4}",
"Problem ", i);
                              Console.WriteLine(" {0}{1,4}",
"Number of cases", n);
                              Console.WriteLine(" {0}{1,4}",
"Number of classes", k);
                              k++;
                              if (iclass != 1)
                              {
                                    Console.WriteLine(" {0}");
                              }
                              else
                              {
                                    sr.Reset();
                                    for (j = 1; j <= k; j++)
                                    {
                                          cb[j - 1] =
double.Parse(sr.Next(), CultureInfo.InvariantCulture);
                                    }
                                    Console.WriteLine(" {0}",
"User-supplied class boundaries");
                              }
                              k = k + 1;
 //
                              G01.g01ae(n, k, x, iclass, cb,
ifreq, out xmin, out xmax, out ifail);
 //
                              Console.WriteLine("");
                              if (ifail == 0)
                              {
                                    Console.WriteLine(" {0}",
"Successful call of g01ae");
                                    Console.WriteLine("");
                                    Console.WriteLine(" {0}",
"*** Frequency distribution ***");
                                    Console.WriteLine("");
                                    Console.WriteLine(" {0}",
" Class Frequency");
                                    Console.WriteLine("");
                                    Console.WriteLine("
{0}{1,8:f2}{2,9}", " \t Up to ", cb[0], ifreq[0]);
                                    k = k - 1;
                                    if (k > 1)
                                    {
```

```
                                                for (j = 2; j <= k;
j++)
                                                {

      Console.WriteLine("\t {0,2:f2}{1,2:f2}{2,2:f2}\t {3,5}",
cb[j - 1 - 1], " to ", cb[j - 1], ifreq[j - 1]);
                                                }
                                          Console.WriteLine("");
                                          }
                                    Console.WriteLine("
{0,8:f2}{1}{2,6}", cb[k - 1], " and over ", ifreq[k + 1 - 1]);
                                    Console.WriteLine("");
                                    Console.WriteLine(" {0}{1,6}",
"Total frequency = ", n);
                                    Console.WriteLine(" {0}{1,9:f2}",
"Minimum = ", xmin);
                                    Console.WriteLine(" {0}{1,9:f2}",
"Maximum = ", xmax);
                                    }
                              else
                              {
                                    Console.WriteLine("** g01ae
failed with ifail = {0,5}", ifail);
                              }
                        }
 //
                  }
            }
                        catch (Exception e)
                        {
                              Console.WriteLine(e.Message);
                              Console.WriteLine( "Exception Raised");
                        }
                  }
            }
}
```

Home: Index

```
@model int[,]
@{
    ViewData["Title"] = "Home Page";
}



    <head>
```

```
        <meta charset=utf-8>
        <style>

            body {
                margin:0;
                padding-top: 75px;
            }

            canvas {
                width: 100%;
                height: 100%;
            }
        </style>
    </head>

    <body>

        <div align="center" class="container">
            @*<canvas id="canvas" width="200" height="200"></canvas>*@
            <p>Скорость движения поддона равна 0,5 метров в секунду</p>
            <p>Частота кадров в секунду равна 50</p>
            <p>Объем всего сырья составляет: <output id="V">0</output> кубических метров</p>
        </div>
        <script src="~/js/three.min.js"></script>
        <script src="~/js/OrbitControls.js"></script>
        <script>
        //const canvas = document.getElementById('canvas');
        var x1 = 300;
        var x2 = 220;
        var y = 140;
        var z = 50;
        let S = new Array();
```

```
        var h1, h2, o1, o2;

        var Model = @Html.Raw(Json.Serialize(Model));

        var renderer = new THREE.WebGLRenderer();
        renderer.setSize(500,375);
        document.body.appendChild(renderer.domElement);

        var renderer3d = new THREE.WebGLRenderer();

renderer3d.setSize(window.innerWidth/2.5,window.innerHeight/1.75
);
        document.body.appendChild(renderer3d.domElement);

        var camera = new THREE.OrthographicCamera(-400, 400, 300,
-300, -1, 0);

        var camera3d = new THREE.PerspectiveCamera(75,
(window.innerWidth/2.5)/(window.innerHeight/1.75), 1,2000);
        camera3d.position.set(0, 0, 500);
        camera3d.lookAt(new THREE.Vector3(0, 0, 0));

        var controls = new THREE.OrbitControls(camera3d,
renderer3d.domElement);
        controls.update();

        var scene = new THREE.Scene();

        var material = new THREE.LineBasicMaterial({ color:
0XFFFFFF }); //белый
        var material2 = new THREE.LineBasicMaterial({ color:
0XFF0000 }); // красный
        var material3 = new THREE.LineBasicMaterial({ color:
0X4D5051 }); // серый

        var group = new THREE.Group();
```

```
        var geometry = new THREE.Geometry();
        geometry.vertices.push(new THREE.Vector3(-x1, y, 0));
        geometry.vertices.push(new THREE.Vector3(-x2, -y, 0));
        geometry.vertices.push(new THREE.Vector3(x2, -y, 0));
        geometry.vertices.push(new THREE.Vector3(x1, y, 0));

        var line = new THREE.Line(geometry, material2);
        scene.add(line)

        var geometry2 = new THREE.Geometry();

        function Sum(array)
        {
            var sum = 0;
            for (var i = 0; i < array.length; i++)
            {
                    sum += array[i];
            }
            return (sum);
        }

        for (var j = 0; j < 2000; j++)
        {
            var Smin = 0;
            var b = true;
            for (var i = 0; i < 600; i++)
            {
                if ((((y - Model[j][i]) / (y*2)) * (x1-x2) < i) &&
(((x1-(-x1)) - (((y - Model[j][i]) / (y*2)) * (x1-x2))) > i) &&
(Model[j][i]>-y))
                {
                    geometry2.vertices.push(new THREE.Vector3(i -
x1, Model[j][i], -j));
                    h2 = Model[j][i] + y;
                    o2 = i - (x1 + x2);
```

```
                    if (i > 0)
                    {
                        Smin = Smin + ((Model[j][i] - Model[j][i
- 1]) * (((i + (i - 1)) / 2) - (i - 1)) + Model[j][i - 1] + y);
                    }
                    if (b == true)
                    {
                        h1 = Model[j][i] + y;
                        o1 = (x1 - x2) - i;
                        b = false;
                    }
                }
            }
            if (((j % z) == 0)&&(j<2050))
            {
                var geometry = new THREE.Geometry();
                geometry.vertices.push(new  THREE.Vector3(-x1,  y,
-j));
                geometry.vertices.push(new THREE.Vector3(-x2, -y,
-j));
                geometry.vertices.push(new  THREE.Vector3(x2,  -y,
-j));
                geometry.vertices.push(new THREE.Vector3(x1, y, -
j));


                geometry.vertices.push(new THREE.Vector3(x1, y, -
j+z));
                geometry.vertices.push(new  THREE.Vector3(x2,  -y,
-j+z));
                geometry.vertices.push(new THREE.Vector3(-x2, -y,
-j+z));
                geometry.vertices.push(new  THREE.Vector3(-x1,  y,
-j+z));
                geometry.vertices.push(new  THREE.Vector3(-x1,  y,
-j));
                geometry.vertices.push(new THREE.Vector3(-x2, -y,
-j));
```

```
                geometry.vertices.push(new THREE.Vector3(-x2, -y,
-j + z));
                geometry.vertices.push(new  THREE.Vector3(x2,  -y,
-j + z));
                geometry.vertices.push(new  THREE.Vector3(x2,  -y,
-j));

                var line = new THREE.Line(geometry, material3);
                group.add(line);
                scene.add(group);
            }

            var lines = new THREE.Line(geometry2, material);
            group.add(lines);
            var geometry2 = new THREE.Geometry();
            S[j] = Smin-0.5*h1*o1-0.5*h2*o2;
        }

        scene.add(group);

        var t = 0;
        function render() {
            t += 1;
            scene.add(group);
            requestAnimationFrame(render);
             group.position.z = 2*t;
            renderer.render(scene, camera);
        }
        render();

        var t2 = 0;
            function render3d()
            {
                t2 += 1;
                requestAnimationFrame(render3d);
```

```
                controls.update();
                 group.position.z = 2*t2;
                renderer3d.render(scene, camera3d);
            }
        render3d();
        var V = Sum(S) / 1000000;
        document.getElementById("V").innerHTML = V;
        </script>
    </body>
```

_ViewImports

```
@using Web
@using Web.Models
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

_ViewStart

```
@{
    Layout = "_Layout";
}
```

Program

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Logging;
```

```
namespace Web
{
    public class Program
    {
        public static void Main(string[] args)
        {
            CreateWebHostBuilder(args).Build().Run();
        }

        public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
            WebHost.CreateDefaultBuilder(args)
                .UseStartup<Startup>();
    }
}
```

Startup

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.HttpsPolicy;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;

namespace Web
{
    public class Startup
    {
```

```
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method
to add services to the container.
        public     void     ConfigureServices(IServiceCollection
services)
        {
            services.Configure<CookiePolicyOptions>(options =>
            {
                // This lambda determines whether user consent for
non-essential cookies is needed for a given request.
                options.CheckConsentNeeded = context => true;
                options.MinimumSameSitePolicy                    =
SameSiteMode.None;
            });

services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.V
ersion_2_2);
        }

        // This method gets called by the runtime. Use this method
to configure the HTTP request pipeline.
        public    void    Configure(IApplicationBuilder    app,
IHostingEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }
            else
            {
```

```
                app.UseExceptionHandler("/Home/Error");
                // The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.
                app.UseHsts();
            }

            app.UseHttpsRedirection();
            app.UseStaticFiles();
            app.UseCookiePolicy();

            app.UseMvc(routes =>
            {
                routes.MapRoute(
                    name: "default",
                    template: "{controller=Home}/{action=Index}/{id?}");
            });
        }
    }
}
```