МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Южно-Уральский государственный университет
(национальный исследовательский университет)»**
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.ф.-м.н.,
профессор

_____ Л.Б. Соколинский

«___»_____ 2024 г.

Моделирование почвы для расчета нагрузок и имитации строительных процессов

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 02.03.02.2024.308-302.ВКР

Научный руководитель,
доцент кафедры СП, к.ф.-м.н.
_____ В.Н. Алеева

Автор работы,
студент группы КЭ-401
_____ Я.А. Козинец

Ученый секретарь
(нормоконтролер)
_____ И.Д. Володченко
«___»_____ 2024 г.

Челябинск, 2024 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

УТВЕРЖДАЮ
Зав. кафедрой СП
_____ Л.Б. Соколинский
29.01.2024 г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы бакалавра
студенту группы КЭ-401
Козинцу Ярославу Алексеевичу,
обучающемуся по направлению
02.03.02 «Фундаментальная информатика и информационные технологии»

1. **Тема работы** (утверждена приказом ректора от 22.04.2024 г. № 764-13/12) Моделирование почвы для расчета нагрузок и имитации строительных процессов.
2. **Срок сдачи студентом законченной работы:** 03.06.2024 г.
3. **Исходные данные к работе**

3.1. Kalachuk T.G., Kara K.A. General calculation principles of structurally-unstable soils deformation. // International scientific and practical conference «Agro-SMART-Smart solutions for agriculture» (Agro-SMART 2018). – Atlantis Press, 2018. – С. 306–310.

3.2. Du J., Huang T. A Method for Traction Ability Research of a Rover Wheel on Mixed Planet Terrain with Movable Stones. // Space: Science & Technology, 2023. – Т. 3. – С. 1–9.

3.3. Kruger R., Els P.S., Hamersma H.A. Experimental investigation of factors affecting the characterisation of soil strength properties using a Bevameter in-situ plate sinkage and shear test apparatus. // Journal of Terramechanics. – 2023. – Т. 109. – С. 45–62.

4. **Перечень подлежащих разработке вопросов**

4.1. Провести обзор аналогов и изучить предметную область.

4.2. Спроектировать и реализовать модель деформируемой поверхности почвы в физическом движке.
4.3. Спроектировать и реализовать гранулированную модель почвы для имитации разрушения почвы в физическом движке.
4.4. Реализовать расчет и вывод нагрузок на взаимодействующие с почвой объекты.
4.5. Провести тестирование модели на основе различных параметров почвы.
5. Дата выдачи задания: 29.01.2024 г.

Научный руководитель,
доцент кафедры СП, к.ф.-м.н. В.Н. Алеева

Задание принял к исполнению Я.А. Козинец

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность

В современном инженерном строительстве и архитектуре особое внимание уделяется разработке технологий, позволяющих более точно моделировать и рассчитывать строительные процессы и нагрузку на строительную технику. Моделирование и симуляция предоставляют инструменты для анализа различных сценариев и вариантов реализации проектов. Это помогает принимать обоснованные решения, учитывать все возможные последствия и выбирать наилучшие стратегии. Одним из важных направлений в этой области является симуляция физической модели почвы, которая позволяет имитировать инженерные процессы на компьютере [1].

Компьютерная симуляция обладает широким спектром применения и рядом преимуществ.

1. Предсказание поведения почвы и техники – позволяет инженерам получить информацию о предстоящем строительном процессе и произвести нужные расчеты.

2. Управление техникой дистанционно – загрузив модель рабочего полигона и технику в движок и подключив его к бортовому компьютеру, можно в онлайн режиме управлять процессом, не садясь за руль самой машины.

3. Машинное обучение – интегрировав нейронную сеть в проект модели почвы, можно обучить ее и создать с помощью нее беспилотную технику.

4. Обучение новых специалистов – подключив к симуляции специальное VR оборудование, можно воссоздать виртуальную кабину управляемой техники и обучать специалистов без задействования строительных машин.

5. Различные инженерные тестирования – могут помочь еще на этапе проектирования новой техники с учетом результатов тестов в физическом движке.

6. Разработка игр – является отдельным направлением, где подобная модель может быть крайне актуальна.

Современные методы моделирования почвы, такие как численное моделирование и использование программного обеспечения на основе конечных элементов, способствуют развитию новых технологий и подходов в строительстве. Это открывает возможности для реализации более сложных и амбициозных проектов, которые раньше были бы слишком рискованными или дорогими.

В целом, перенос любого рабочего процесса в компьютерную симуляцию является большим шагом к упрощению и автоматизации этого процесса, позволяя заменить сложно-восполняемые ресурсы на компьютерные.

Постановка задачи

Целью выпускной квалификационной работы является моделирование почвы для расчета нагрузок и имитации строительных процессов. Для достижения поставленной цели необходимо решить следующие задачи:

1) провести обзор аналогов и изучить предметную область;

2) спроектировать и реализовать модель деформируемой поверхности почвы в физическом движке;

3) спроектировать и реализовать гранулированную модель почвы для имитации разрушения почвы в физическом движке;

4) реализовать расчет и вывод нагрузок на взаимодействующие с почвой объекты;

5) провести тестирование модели на основе изменения различных параметров почвы.

Структура и содержание работы

Работа состоит из введения, четырех глав, заключения, списка литературы и приложения. Объем работы составляет 49 страниц, объем списка литературы – 16 источников.

В первой главе описывается предметная область разработки физических симуляций почвы и строительных процессов. Также приводится анализ имеющихся моделей и симуляторов почвы. Кроме этого, приводится сравнительный анализ физических движков с целью выбора оптимального для использования в проекте.

Вторая глава посвящена проектированию деформируемой поверхности, физической модели почвы и гранул. В ходе работы над данной главой были определены варианты использования, функциональные и нефункциональные требования. Спроектированы модель поверхности, физика взаимодействия твердых тел с почвой, физика взаимодействия гранул друг с другом и почвой.

В третьей главе представлена реализация разрабатываемой модели. Глава делится на реализацию деформируемой поверхности почвы, реализацию модели создания гранул, расчет и визуализацию нагрузок. В данной главе также приведены листинги кодов разрабатываемой программы.

Четвертая глава посвящена тестированию, в котором разработанная модель проходит проверку на соответствие ее физическим параметрам почвы.

В приложении А представлена диаграмма деятельности работы физической модели почвы.

В приложении Б представлены листинги реализации вычисления поверхностей объектов, взаимодействующих с почвой.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Описание предметной области

В инженерной сфере существует острая потребность в точном и реалистичном моделировании воздействия нагрузок со стороны почвы и ее изменения в результате применения к ней механических процессов. Физические и инженерные аспекты взаимодействия с почвой становятся ключевыми факторами в проектировании и строительстве. Одно из главных преимуществ физических симуляций – отсутствие необходимости задействовать технику и человеческие ресурсы для прогнозирования и управления строительным процессом.

Большинство инженерных компаний на данный момент занимаются наймом высококвалифицированных специалистов, умеющих управлять новейшей строительной техникой. Но с ростом прогресса в совершенствовании техники таких специалистов становится все меньше, а компаниям становится невыгодным использование дорогого человеческого труда. На помощь им приходят компьютерные симуляции, позволяющие не только обучать специалистов, но и прогнозировать строительные процессы с учетом всевозможных нагрузок на технику, а в перспективе оснащать бортовые компьютеры программами для дистанционного управления.

В данный момент на рынке не существует универсальных продуктов с постоянной поддержкой, которые бы удовлетворяли потребности инженеров и работодателей. Основной проблемой является сложность физических расчетов в real-time симуляции. Поэтому большинство имеющихся симуляторов используют модель на основе предварительного сценария, не позволяющую управлять ею в режиме выполнения. Например, как физический движок Chrono [2].

Для того, чтобы добиться желаемого результата и получить оптимальную производительность, было принято использовать игровой движок, т.к. в отличие от обычных фреймворков для сложных физических симуляций, игровые движки направлены на real-time моделирование.

Важными аспектами выбираемого движка должны быть: производительность, его способность интегрировать в себя другие модули и фреймворки, способность самому интегрироваться, стабильность симуляции, кроссплатформенность, поддержка VR.

После проведения анализа рынка игровых движков были выбраны следующие продукты, обладающие постоянной поддержкой разработчиков и новейшими дополнениями в 3D разработке: Unreal Engine [3], Unity [4], Godot [5], Havok Physics [6], Cry Engine [7].

Для каждого из них были выделены преимущества и недостатки в имеющих важность для проекта аспектах. Результаты анализа представлены в таблице 1.

Таблица 1 – Сравнительные характеристики игровых движков

Игровой движок	**Производительность**	**Интеграция**	**Кроссплатформенность**	**Инструментарий**	**Требовательность к ресурсам**
Unity	Средняя	Широкая	Широкая	Широкий	Средняя
Unreal Engine	Средняя	Широкая	Ограниченная	Широкий	Высокая
Godot	Высокая	Ограниченная	Ограниченная	Узкий	Низкая
Havok Physics	Высокая	Широкая	Ограниченная	Узкий	Средняя
Cry Engine	Высокая	Ограниченная	Ограниченная	Узкий	Высокая

По результатам сравнения стало ясно, что наиболее подходящим является движок Unity, который обладает широким спектром инструментов, легко интегрируется, позволяет создавать приложения для различных платформ и имеет поддержку VR.

1.2. Сравнительный анализ аналогов

На рынке представлен очень небольшой ассортимент, удовлетворяющий требованиям поставленной задачи. Самым популярным и продвинутым

инженерным решением является AGX Dynamics для Unreal Engine от Algoryx.

Algoryx – мировой лидер в области моделирования физики в виртуальных средах, используемых в передовых инженерных разработках. Ее флагманское решение, AGX Dynamics, представляет собой технологию для инженеров и дизайнеров в самых разных отраслях: от горнодобывающей, лесной и морской до аэрокосмической, робототехники, строительства и многих других. Их технологии позволяют моделировать сложные физические явления и процессы, что находит применение в разработке и тестировании робототехнических систем, симуляции морских и горнодобывающих операций, а также в создании программных продуктов для авиационной и оборонной отраслей. Интеграция AGX Dynamics в Unreal Engine предоставляет им высокоточные физические симуляции в 3D.

Основным функциональным компонентом Algoryx является симулятор взаимодействия с почвой, позволяющий тестировать строительную технику внутри движка Unreal. AGX Soil – это симулятор, разработанный компанией AGX для моделирования и анализа поведения почвы в различных условиях. С помощью AGX Soil можно моделировать воздействие различных нагрузок на почву, чтобы предсказать ее реакцию и деформации в ответ на эти нагрузки. Программное обеспечение AGX Soil широко применяется в сельском хозяйстве для оптимизации процессов обработки почвы, планирования посевных работ и увеличения урожайности. Оно также используется в строительстве и геологии для анализа свойств грунта, прогнозирования деформаций при строительстве и проведения других исследований. На рисунке 1 представлены скриншоты тренировочного полигона на основе AGX Terrain, симулирующего почву.

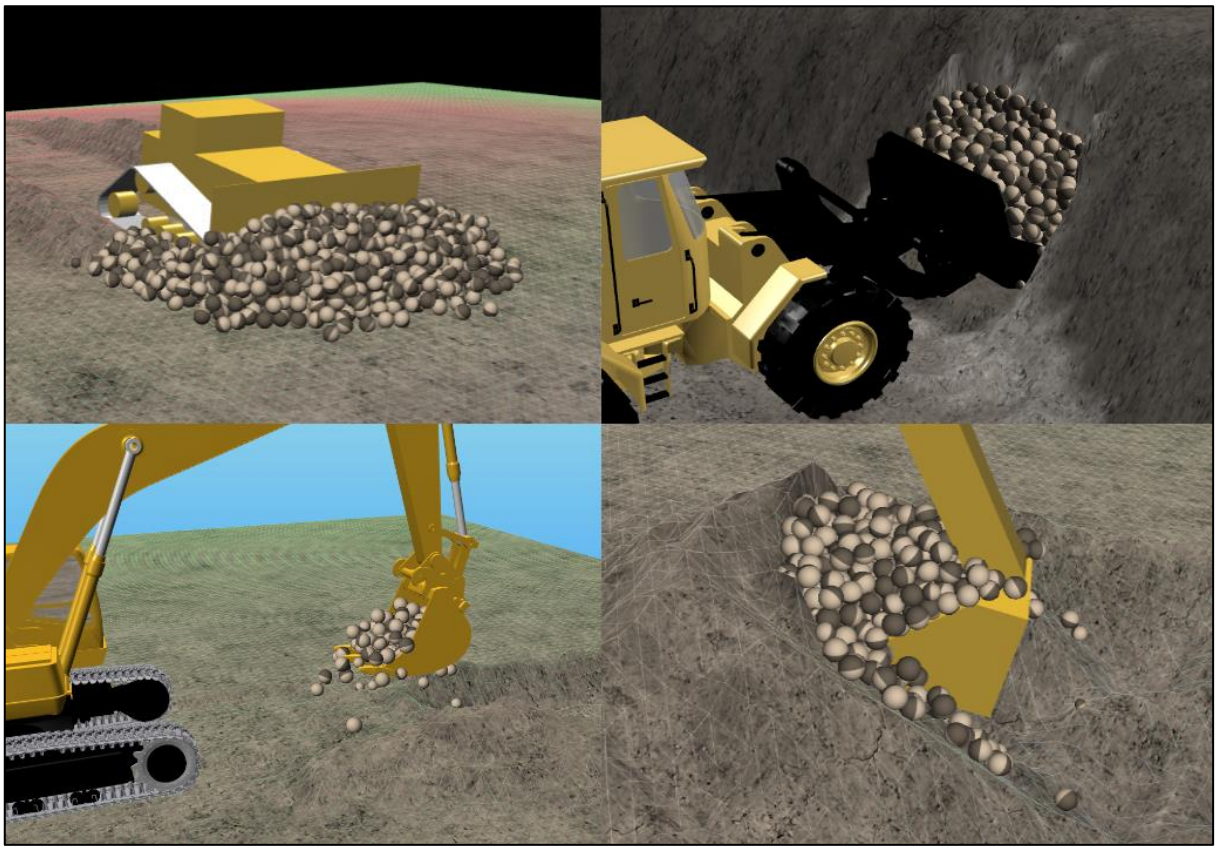

Рисунок 1 – Скриншоты AGX Terrain от Algoryx

Подобное решение на основе AGX обладает следующими функциональными возможностями:

1) физическое взаимодействие твердых тел с почвой;
2) деформация почвы на основе приложенных к ней сил;
3) эрозия почвы;
4) преобразование сеточной модели в гранулы, позволяющие копать почву;
5) настройка параметров почвы под конкретный ее тип;
6) свободная работа с полигоном в real-time времени.

Одни из главных минусов AGX – отсутствие доступа к продуктам в России и суженные возможности редактирования симуляции под пользовательские нужды, а также отсутствие возможности отслеживать нагрузки на технику. Полный функционал к своему продукту AGX предоставляет только компаниям, готовым платить большие деньги за использование их

продукта. Для инженеров крайне важно иметь широкий спектр инструментов для управления симуляцией, в частности настраивать почву и ее поведение под необходимые нужды, а также возможность аппроксимировать модель. Большинство развивающихся компаний не готово платить деньги за данный продукт. Кроме того, достойных отечественных аналогов на рынке не существует, и инженерные решения в данной области имеют множество ограничений, что говорит об актуальности наличия подобного продукта на российском рынке.

Таким образом, можно сделать вывод, что реализуемая модель симулятора должна обладать широкими возможностями редактирования ее параметров и стабильную работу в real-time режиме с учетом реальных физических расчетов, а также осуществлять расчет и вывод нагрузок на взаимодействующие с ней объекты.

Вывод по первой главе

В ходе анализа предметной области был выбран движок для дальнейшей разработки симулятора и изучены существующие аналоги. Были определены актуальные аспекты разрабатываемого проекта, что подчеркнуло его значимость и необходимость. Обзор аналогичных решений помог выделить ключевые функции моделируемой системы, такие как физическая достоверность и интерактивность. Также были рассмотрены технические и пользовательские требования, что позволило выделить основной функционал разрабатываемой модели, чтобы перейти к детальному анализу требований.

2. ПРОЕКТИРОВАНИЕ

2.1. Анализ требований

В рамках работы были выявлены функциональные требования к системе, которые определяют поведение системы во время выполнения симуляции вне зависимости от ее реализации. Для разрабатываемой деформируемой поверхности почвы выявлены следующие функциональные требования:

1) почва должна деформироваться в соответствии с формой и физическими параметрами объекта, оказывающего на нее силу;

2) почва должна быть масштабируемой в соответствии с параметрами пользователя;

3) почва должна вести себя подобно твердому телу, взаимодействуя с другими объектами;

4) деформация почвы должна происходить в реальном времени;

5) деформация и разрушение почвы должны осуществляться на основе параметров, настраиваемых пользователем.

Также с учетом некоторых особенностей, которым должна соответствовать физическая симуляция, были выявлены следующие нефункциональные требования:

1) программная оболочка системы должна быть реализована на языке программирования C# для определения ее движком Unity;

2) конфигурация деформируемой поверхности должна настраиваться самим пользователем.

2.2. Диаграмма вариантов использования

Для определения необходимого функционала симулятора модели почвы была разработана диаграмма вариантов использования, отображающая взаимодействие пользователя с программой. Диаграмма приведена на рисунке 2.

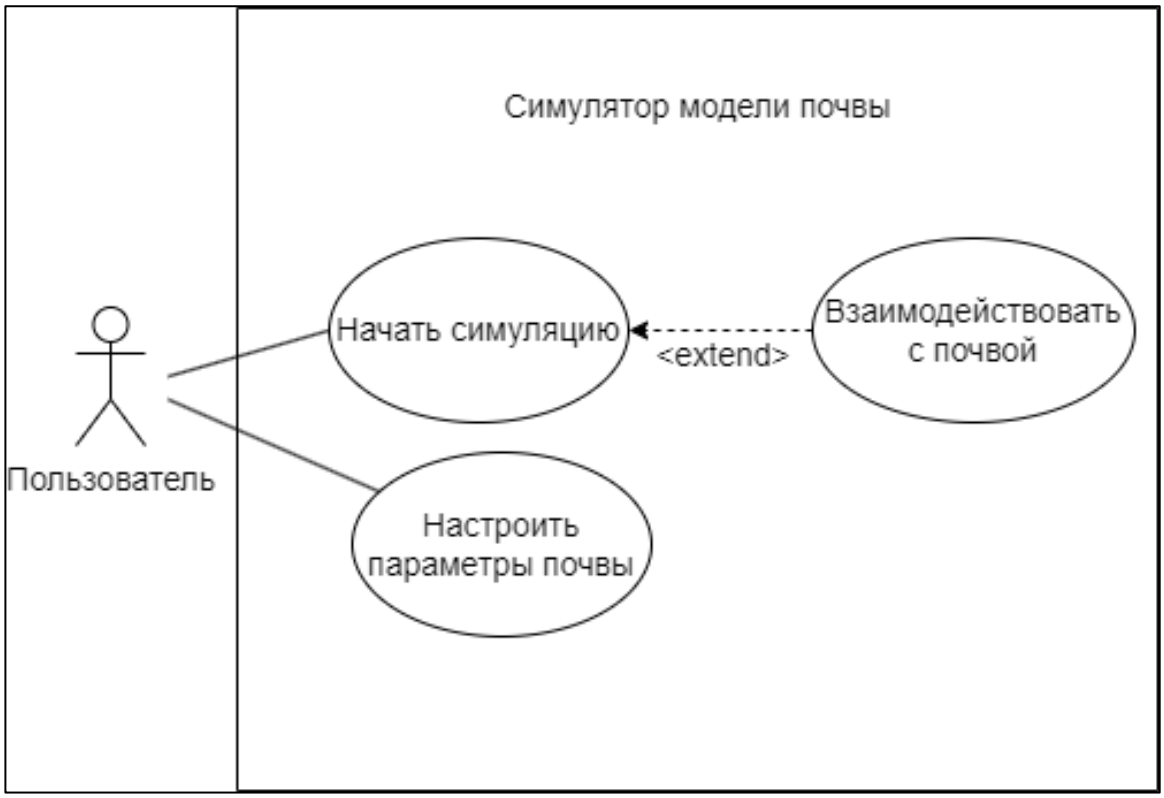

Рисунок 2 – Диаграмма вариантов использования симулятора почвы

С системой взаимодействует один актер – пользователь, который может запускать симуляцию и настраивать параметры почвы. В режиме симуляции пользователь имеет возможность управлять предварительно настроенной моделью техники, взаимодействуя с почвой и осуществлять копательные процессы.

2.3. Алгоритм работы модели почвы

По результатам анализа предметной области и требований к разрабатываемой модели было принято решение использовать гибридный подход к симуляции модели почвы. Подход заключается в совмещении сетчатой модели вершин, образующую поверхность почвы, и метода дискретных элементов (DEM) [8], моделирующего гранулы почвы. В гибридном подходе для имитации уплотнения почвы используется деформируемая поверхность, обладающая определенными параметрами, задающими поведение почвы.

Для имитации разрушения и копательных процессов используется гранулированный подход, где частички земли представлены в виде отдельных гранул. На рисунке 3 представлена блок-схема алгоритма, реализующего данный подход в физическом движке.

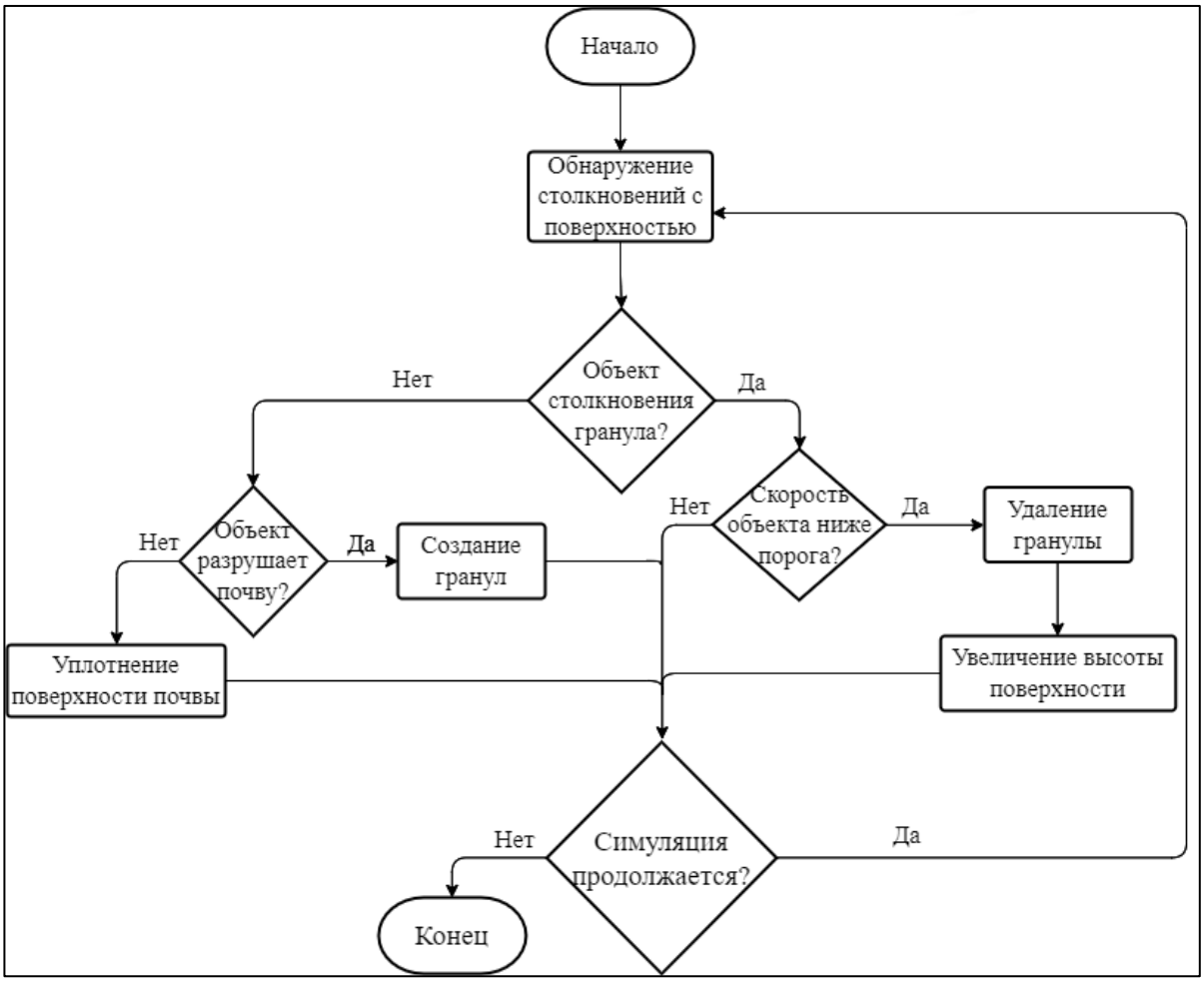

Рисунок 3 – Блок-схема алгоритма работы модели почвы

После запуска симуляции система обнаружения столкновений движка отслеживает с какими объектами происходит столкновение почвы. В случае, если объект не является частью земли, требуется узнать произошло ли разрушение почвы после взаимодействия объекта с ней. Если нет, поверхность почвы уплотняется, в противном случае создаются гранулы почвы. Если же почва сталкивается с гранулой, необходимо узнать ее скорость. В случае, когда скорость достаточно мала, гранулу необходимо удалить, а на ее месте

поднять высоты вершин поверхности. Данный алгоритм продолжает свою работу до тех пор, пока симуляция активна.

2.4. Архитектура деформируемой поверхности

В подавляющем числе физических движков, как и в Unity, объекты на сцене генерируются на основе сетчатой модели, представляющей из себя поверхность связанных между собой вершин. В качестве основного такого объекта выступает Mesh [9] с компонентом Mesh Filter, включающим в себя обработчик сетки вершин. Более удобным и модернизированным вариантом сетчатой поверхности в Unity является Terrain [10], имеющий множество внутренних методов работы с картой высот и параметрами вершин, а также использующийся непосредственно для моделирования статичной плоскости, выступающей в роли земли в игровых проектах. Поэтому именно Terrain и был выбран за основу деформируемой поверхности, которая в будущем будет представлять из себя динамическую почву. На рисунке 4 представлен пример объекта Terrain с настроенной картой высот.

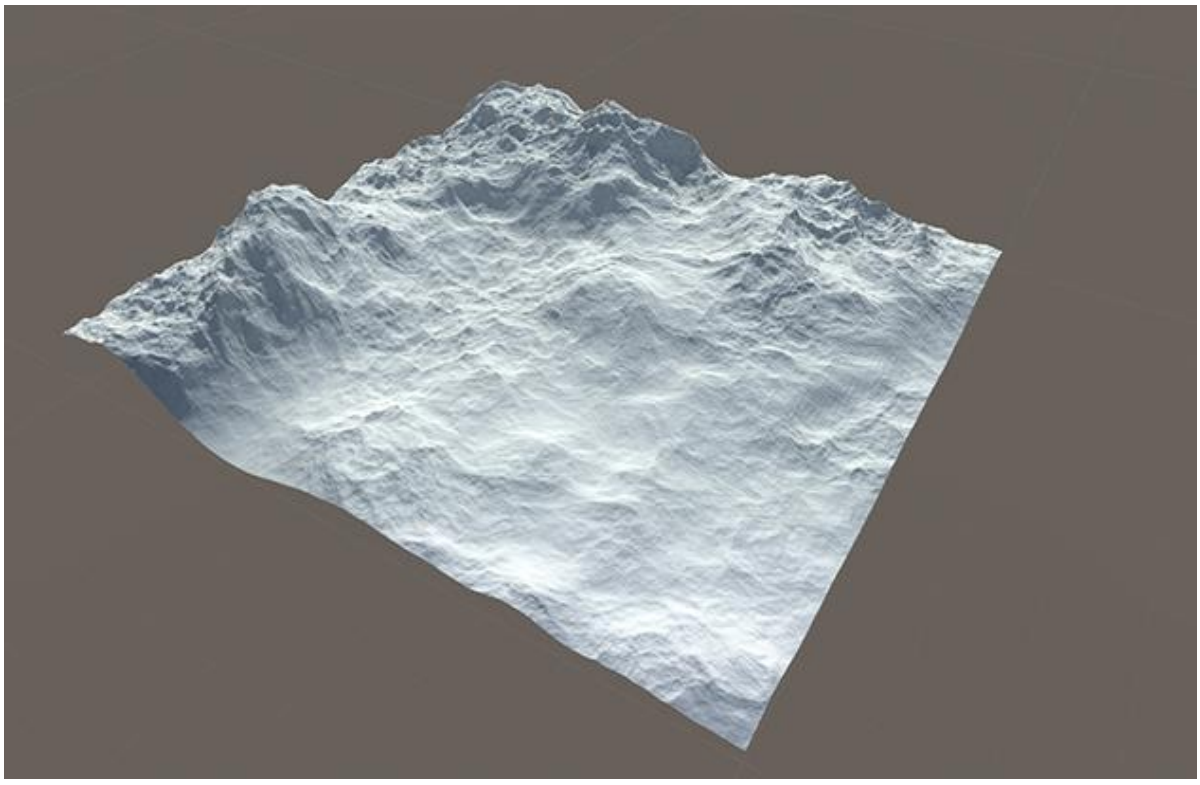

Рисунок 4 – Terrain в Unity

2.4.1. Отслеживание точек касаний

Для того чтобы деформировать поверхность в соответствии с действующими на нее силами от твердого тела в первую очередь необходимо найти точки, в которых происходит касание объекта с поверхностью. В Unity есть встроенная система коллизий для отслеживания точек контактов между телами, однако данный метод не подходит для проектируемой модели, так как симулируемая почва должна обладать возможностью допускать проникновение объектов в себя, а также рассчитывать тангенциальные силы, которые при обычном контакте найти становится затруднительно. Наиболее подходящим вариантом будет использование поверхности, не участвующей в системе коллизий физического движка с взаимодействующими объектами [11]. Поэтому в качестве отслеживающих компонентов будет выступать система лучей RayCast [12], которая обрабатывается самим движком и позволяет бросать векторные лучи из определенной точки, чтобы получать информацию о задетом объекте. На рисунке 5 представлена схема бросания луча, где Ray – луч, Origin – точка начала луча, Direction – направление луча, Hit – точка попадания луча в объект.

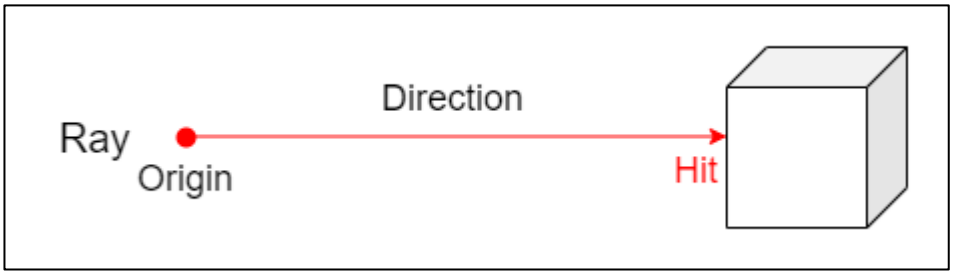

Рисунок 5 – Схема Raycast в Unity

В деформируемой модели почвы данная система работает по следующему алгоритму:

1) отслеживается позиция объекта, участвующего в деформационной системе;

2) на основе позиции объекта создается область прямоугольная x на y, значения которой регулируются пользователем;

3) в заданной области под поверхностью создаются лучи с шагом n, который также регулируется извне;

4) на каждом шаге симуляции осуществляется проверка лучей на попадание в объект;

5) если луч попал, считается, что данная точка на поверхности является точкой касания.

Таким образом, на каждом шаге симуляции происходит проверка на точку контакта поверхности с объектом, участвующим в симуляции.

2.4.2. Деформация поверхности

Следующим шагом моделирования деформации является смещение вершины поверхности с учетом взаимодействующего с ней объекта. Для реализации данного процесса используется информация об объекте из удара луча RayCast следующим образом:

1) для каждого луча рассматриваются четыре соседних вершины;

2) из них определяются те, которые участвовали в контакте, то есть те, которые были задеты другим соседним лучом;

3) для каждого луча по подобному принципу заполняется структура, хранящая позиции соседних контактов;

4) заполненная структура формирует связь, определяющая собой контактную область из максимум пяти связанных между собой вершин;

5) для каждой связи считается площадь и периметр выпуклой области, это необходимо для дальнейшего расчета сил и нагрузок.

Для расчета области контакта был выбран алгоритм Джарвиса, работающий по следующему принципу.

1. Рассматривается точка p_0 текущего множества с самой маленькой y-координатой (если таких несколько, берется самая правая из них). Точка добавляется в ответ.

2. На каждом следующем шаге для последнего добавленного p_i находится p_{i+1} среди всех не добавленных точек и p_0 с максимальным полярным углом относительно p_i (если углы равны, надо сравнивать по расстоянию). Точка p_{i+1} добавляется в ответ.

3. Если p_{i+1} равна p_0, алгоритм завершается.

Помимо расчета области контакта, необходимо произвести расчет высоты погружения объекта в поверхность: от высоты вершины точки касания отнимается высота точки попадания лучом в объект. На основе этих данных и параметров почвы в дальнейшем будет производиться физический расчет высоты деформации почвы. На данном этапе достаточно сместить вершины на величину погружения объекта в поверхность.

2.4.3. Применение сил к объекту

Чтобы ограничить деформацию, необходимо применить силы к объекту в каждой точке касания, вытеснив его на верхнюю плоскость поверхности. В следующей главе описан расчет сил на основе параметров почвы и нормальных и тангенциальных сил, действующих на объект.

Также, чтобы объект мог физически взаимодействовать с почвой, например, катиться по ней, используются два альтернативных похода для каждого из предпочитаемых пользователем типов симуляции.

Первый из них – модель абстрактной физики, в которой все скорости передаются объекту не через внутренние методы управления твердым телом движком, а напрямую, в виде изменения позиции и вращения объекта, благодаря чему освобождаются вычислительные ресурсы симуляции, а тело получает возможность передвигаться по поверхности.

Второй подход – модель создания дополнительных точек опоры, прикрепленных к объекту, которые участвуют в физическом взаимодействии с поверхностью, обрабатываясь самим движком, но не участвуют в общей модели RayCast и расчете сил.

Каждая из моделей определяется предварительной настройкой объекта, представляющего рабочую технику, участвующую в симуляции. Скрипты, реализующие поведение техники, пишутся пользователем самостоятельно. В разрабатываемом проекте используется второй вариант, так как не требует создания дополнительных скриптов и реализации механик перемещения.

2.5. Физическая модель почвы

В качестве физической модели для почвы была выбрана модель Беккера [13]. Модель Беккера представляет собой метод, который позволяет описать поведение почвы при контакте с твердым объектом. Этот метод обеспечивает расчет сил, действующих на движущийся объект, и основан на ряде параметров, таких как коэффициенты сцепления, пластичности и упругости.

Модель Беккера использует следующие переменные:

1) Bekker_Kphi – коэффициент трения почвы (выражается в [Па/м]);
2) Bekker_Kc – коэффициент сцепления почвы (выражается в [Па/м]);
3) Bekker_n – экспоненциальный коэффициент модели;
4) Mohr_cohesion – коэффициент пластичного сцепления Мора (выражается в [Па];
5) Mohr_mu – угол трения Мора (выражается в градусах);
6) Janosi_shear – коэффициент Яноши-Ханамото, определяющий изменение касательных напряжений (выражается в [м]);
7) Elastic_K – коэффициент упругости почвы (выражается в [Па/м]);
8) Damping_R – коэффициент демпфирования, то есть вертикального затухания почвы (выражается в [Па*с/м]).

Данные параметры определяют тип почвы и ее реакцию на взаимодействие с объектами и могут регулироваться пользователем для достижения необходимого поведения почвы.

В расчетах сил по модели Беккера используется модель Мора [14], основанная на расчете максимального касательного напряжения, угла внутреннего трения и общих сил трения. Модель Беккера также позволяет сразу вычислить нагрузку, действующую на объект, который взаимодействует с почвой в конкретной точке.

Алгоритм расчета сил почвы на объект основан на следующих шагах:

1) определение точки контакта объекта с почвой;

2) расчет вертикального давления почвы на объект (нормальной силы);

3) расчет тангенциального напряжения вдоль поверхности контакта (касательной силы);

4) учет пластической деформации почвы;

5) расчет силы трения по модели Мора;

6) расчет суммарной силы, действующей на объект.

Работу алгоритма можно описать следующим образом:

1) вычисление вертикального давления на объект, исходя из его погружения в почву и коэффициентов упругости и сцепления почвы;

2) если вертикальное давление отрицательное, оно приравнивается к нулю, поскольку почва не может оказывать отталкивающую силу;

3) расчет касательной деформации почвы и ее аккумуляции во времени;

4) коррекция пластической деформации при достижении предельного напряжения;

5) расчет максимального касательного напряжения по модели Мора и учет его в формировании касательной силы;

6) вычисление суммарной силы, учитывающей как нормальное, так и касательное давление.

На рисунке 1 в приложении А приведена диаграмма деятельности работы физической модели почвы на одном шаге симуляции.

Сила, действующая на объект со стороны почвы вычисляется по формуле (1):

$$\vec{F} = \overrightarrow{F_n} + \overrightarrow{F_t}\,, \tag{1}$$

где $\overrightarrow{F_n}$ – нормальная сила со стороны почвы;

$\overrightarrow{F_t}$ – тангенциальная сила со стороны почвы.

Нормальная сила вычисляется по формуле (2):

$$\overrightarrow{F_n} = \vec{N} * S_c * \sigma\,, \tag{2}$$

где $\vec{N}$ – нормаль к текущей точке на сетке вершин;

S_c – площадь одной клетки на сетке вершин;

σ – напряжение в текущей точки связи, определяемое величиной погружения, расчет которой приведен в формуле (3):

$$\sigma = \left(Bekker_{Kc} * OOB\ + Bekker_{Kphi}\right) * sinkage^{Bekker_n}. \tag{3}$$

В данной формуле помимо параметров почвы используется OOB – область контакта, определяющаяся отношением периметра к двум площадям области, $sinkage$ – вычисленное на данном шаге погружение точки объекта в узле.

Тангенциальная сила вычисляется по формуле (4):

$$\overrightarrow{F_t} = \vec{T} * S_c * \tau\,, \tag{4}$$

где $\vec{T}$ – тангенциальное направление, рассчитываемое по формуле (5):

$$\vec{T} = -(\vec{V_o} - (\vec{V_n} * \vec{N})\,, \tag{5}$$

где $\vec{V_o}$ – скорость объекта в данной точке касания с поверхностью;

$\vec{V_n}$ – скалярное произведение векторов нормали и скорости объекта.

В формуле (4) τ – касательное напряжение, рассчитываемое по формуле (6):

$$\tau = (Mohr_{cohesion} + \sigma * Mohr_{mu}) * \left(1 - e^{\frac{kshear}{Janosi_{shear}}}\right), \tag{6}$$

где $kshear$ – переменная, накапливающая касательную деформацию, определяющаяся скалярным произведение скорости и отрицательным тангенциальным направлением за шаг симуляции.

Таким образом, суммирующая сила $\vec{F}$ будет действовать на объект со стороны почвы, выталкивая его на свою границу, а также выступать показателем нагрузки в данной точке объекта. Сумма этих сил – есть общая нагрузка на объект в текущий момент.

2.6. Физическая модель гранул

2.6.1. Определение момента создания гранул

Для создания гранул и моделирования процесса выкапывания почвы в первую очередь необходимо определить моменты создания гранул, представляющих собой выкапываемый материал почвы. Для этого пользователь может задать константный параметр $flowFactor$, определяющий предельный коэффициент разрушения. Он позволит различать, когда почва прминается, а когда разрушается. Гранулы будут создаваться именно в момент разрушения почвы. Данный коэффициент будет сравниваться с текущим значением пластической нагрузки в узле сетки деформируемой поверхности. Вычисление текущего порога разрушения $threshold$ описано в формуле (7):

$$threshold = \eta * sinkage\,, \tag{7}$$

где $sinkage$ – вычисленное раннее на данном погружение точки объекта в узле;

η – шаг пластического течения.

Значение η вычисляется на каждом шаге симуляции с учетом значения на предыдущем шаге по формуле (8):

$$\eta = \frac{sinkage_{plastic} - sinkage_{plastic}{}^{*}}{\Delta t}, \tag{8}$$

где $sinkage_{plastic}{}^{*}$ – пластическая деформация почвы в узле стеки, вычисленная на предыдущем шаге симуляции;

Δt – временной шаг симуляции;

$sinkage_{plastic}$ – текущая пластическая деформация почвы в узле сетки, вычисляемая по формуле (9):

$$sinkage_{plastic} = sinkage\ - \frac{\sigma}{Elastic_K}. \quad (9)$$

В данной формуле участвуют уже рассчитанные ранее $sinkage$ и σ, а также коэффициент упругости почвы $Elastic\_K$.

Таким образом, если рассчитанный на данном шаге $threshold$ больше $flowFactor$, текущий момент деформации в узле сетки является моментом разрушения почвы.

2.6.2. Определение размера и позиции создаваемых гранул

Определение объема гранул

Следующим шагом создания гранул является определение количества выкапываемого материала и направление, в котором он будет создаваться на сцене. Для более оптимизированного моделирования размеры и масса гранул будут фиксированы, а их количество будет зависеть от количества разрушенного материала. Зная погружение объекта в текущем узле и площадь клетки можно найти объем разрушенного материала по формуле (10):

$$V = sinkage * S_c/3\,. \quad (10)$$

В данной формуле во избежание сложных расчетов симуляции форма разрушенного материала принимается за пирамиду. Это допущение разумно, так как в Unity сетка вершин строится из треугольников, соединяющихся в прямоугольник, S_c– площадь одного прямоугольника, а $sinkage$ – высота погружения относительно горизонтальной плоскости почвы. На основе объема также вычисляется масса гранул, путем умножения объема на плотность материала.

Определение позиции гранул

Последним шагом является определение позиции созданных гранул. Во-избежание ситуации, когда гранула создается внутри объекта позицией

создания ее создания принимается следующая точка сетки вершин в направлении скорости объекта.

Однако так гранулы могут создаваться дальше, чем объект фактически выкопал почву. Исправить это можно, используя скорость объекта для прогнозирования следующего шага симуляции, на котором объект будет однозначно граничить с уже созданной гранулой. Для такого определения позиции создания гранулы используется формула (11):

$$\vec{P} = (P_{hitX} + V_{dirX}, h_{nextY} + 2 * V, P_{hitZ} + V_{dirZ}), \quad (11)$$

где P_{hitX} и P_{hitZ} – координаты x и z точки касания почвы с объектом;
V_{dirX} и V_{dirZ} – координаты x и z нормализованного вектора скорости относительно нормали текущей вершины сетки;
h_{nextY} – высота следующей вершины сетки в направлении скорости объекта.

Добавочное значение высоты $2 * V$ используется для однозначного создания гранулы выше поверхности сетки.

После этих вычислений можно создать n гранул в текущей точке, где n определяется отношением объема выкапываемого материала к константному значению объема одной гранулы. Гранулы в этой точке будут создаваться по высоте нормали вершины с шагом V.

2.6.3. Физическое взаимодействие гранул

Созданные гранулы, имитирующие выкапываемую почву, должны обладать физическими свойствами, основанными на параметрах почвы. В симуляции их взаимодействие определяет сила трения, коэффициент трения которой определяется величиной параметра $Mohr_{mu}$ почвы, а также сила сцепления, определяемая коэффициентом сцепления $Mohr_{cohesion}$.

Для моделирования поведения гранул не как отдельных частиц, а потока почвы, используется закон сохранения импульса с учетом коэффициента упругости частиц, учитывающий упругость столкновения. Коэффициент упругости вычисляет на основе параметров почвы по формуле (12):

$$\mu = Mohr_{cohesion} * Janosi_{shear}\,. \quad (12)$$

Узнать значение векторов скоростей для гранул можно исходя из закона сохранения импульса с учетом коэффициента упругости [15]. Формулы (13) и (14) описывают вычисление новых скоростей для каждой из столкнувшихся гранул:

$$\overrightarrow{V_{1^*}} = \overrightarrow{V_1} + (1+\mu) * \frac{m_2}{m_1 + m_2} * \overrightarrow{V_3}\,, \quad (13)$$

$$\overrightarrow{V_{2^*}} = \mu \frac{m_1}{m_1 + m_2} * \overrightarrow{V_3}\,, \quad (14)$$

где $\overrightarrow{V_{1^*}}$ и $\overrightarrow{V_{2^*}}$ – новые скорости объекта после столкновения;

$\overrightarrow{V_3}$ – начальная относительная скорость объектов.

Также на частицы должна действовать сила трения, основанная на коэффициенте трения в модели Беккера. Расчет силы трения осуществляется по формуле (15):

$$\overrightarrow{F_т} = -Mohr_{mu} * \hat{n} * F_{friction}, \quad (15)$$

где $Mohr_{mu}$ – коэффициент трения;

$\hat{n}$ – нормализованный вектор нормали точки контакта;

$F_{friction}$ – величина силы трения, равная величине вектора нормали.

Также необходимо вычислить нагрузку сил, действующих от гранул к другому объекту. Для этого используется третий закон Ньютона, с учетом закона всемирного тяготения по формуле (16):

$$\overrightarrow{F_g} = -G * \frac{m_1 * m_2}{r^2}, \quad (16)$$

где G – гравитационная постоянная;

r – расстояние между центрами тяжести объектов.

Вывод по второй главе

С учетом всех функциональных и нефункциональных требований была разработана модель деформируемой поверхности, работающая на основе системы бросания лучей и применения сил к объекту, а также модель гранул, представляющих из себя разрушенную почву.

3. РЕАЛИЗАЦИЯ

3.1. Реализация деформируемой поверхности

Для реализации сформированной архитектуры использовалась стандартная сцена движка Unity, которая представляет собой область симуляции. На сцене был создан основной объект для представления деформируемой поверхности – Terrain, к которому был прикреплен скрипт на языке C#, реализующий деформацию.

В первую очередь в созданном классе для объекта Terrain на каждом шаге симуляции осуществляется инициализация координат отслеживаемого на сцене объекта, после чего в цикле создаются лучи в прямоугольной области x на y под плоскостью поверхности.

3.1.1. Создание лучей

Для того, чтобы определить точку создания луча, требуется перевести координаты точек высоты объекта в локальную систему координат поверхности. Для этого используется функция `Get`, которая с учетом позиции объекта возвращает текущую высоту поверхности в заданных x и y. Код функции приведен в листинге 1.

Листинг 1 – Функция для определения высоты Terrain в точках x и y

```
public float Get(float x, float z)
{
   return Get((int)x, (int)z);
}
public float Get(int x, int z)
{
   x = (x + heightmap_width) % heightmap_width;
   z = (z + heightmap_height) % heightmap_height;
   return heightmap_data[z, x] * terrain_data.heightmapScale.y;
}
```

Лучи создаются с помощью встроенной в Unity структуры `RaycastHit` для каждой точки прямоугольной области x на y. После того как лучи проинициализированы, они способны хранить информацию о задетой точки объекта. Если в ходе симуляции объект задет лучом – считается что данная точка является точкой касания объекта и поверхности.

Далее для каждого луча осуществляется цикл проверки его на связность с соседними лучами для определения пятна контакта. Данные записываются в структуру `NodeRecord`, хранящую информацию о луче и задетом объекте. После того, как все пятна контактов собраны, осуществляется расчет их площади и периметра. В листинге 2 представлена реализация части кода, в котором происходит заполнение массива структур `NodeRecord`.

Листинг 2 – Заполнение массива точек касаний

```
if (!m_grid_map.ContainsKey(ij))
{
   var normal = terrain.GetNormal(ij);
   var Y = terrain.Get(x + xi, z + zi);
   nodeRecord = new NodeRecord(Y, Y, normal);
   m_grid_map.Add(ij, nodeRecord);
}
counterHitsSphere++;
HitRecord record = new HitRecord();
record.contactable = sphereHit.collider.attachedRigidbody;
record.abs_point = sphereHit.point;
record.patch_id = -1;
hitPositionsSphere.Add(ij, record);
```

В данном коде заполняется словарь `m_grid_map`, где в качестве ключа выступает вектор координат x и z точки попадания луча, а в качестве значения объект структуры `NodeRecord`, которая хранит все физические значения для данной точки. На данном этапе сохраняются только высота точки и нормаль к ней. Параллельно заполняется словарь `HitRecord`, хранящий информации об объекте и точки касания.

После этого необходимо определить какие из точек касания соединяются друг с другом и образуют пятна контакта. В листинге 3 представлен цикл, заполняющий структуру `ContactPatchRecord`, в которой хранятся данные о текущей компоненте связи.

Листинг 3 – Поиск связанных точек и заполнение структуры пятна контакта

```
if (hitPositionsSphere.Count != 0)
{
   var first = hitPositionsSphere.First();
   HitRecord hitRecord = first.Value;
   hitRecord.patch_id = numContactPatches++;
   Vector2 recordKey = first.Key;
   hitPositionsSphere[recordKey] = hitRecord;
   contactPatch.nodes.Add(recordKey);
   Vector2 globalIj = terrain.Grid2World(recordKey);
```

```
    contactPatch.points.Add(globalIj);
    Queue<Vector2> todoQueue = new Queue<Vector2>();
    todoQueue.Enqueue(first.Key);
    while (todoQueue.TryDequeue(out Vector2 cycleTodo))
    {
        int crtPatchId = hitPositionsSphere[cycleTodo].patch_id;
        for (int k = 0; k < 4; k++)
        {
            Vector2 nbrIj = cycleTodo + neighbors4[k];
            if (!hitPositionsSphere.ContainsKey(nbrIj))continue;
            if (hitPositionsSphere[nbrIj].patch_id != -1)continue;
            if (hitPositionsSphere.TryGetValue(nbrIj, out HitRecord value))
            {
                value.patch_id = crtPatchId;
            }
            hitPositionsSphere[nbrIj] = value;
            contactPatch.nodes.Add(nbrIj);
            Vector2 globalNbrij = terrain.Grid2World(nbrIj);
            contactPatch.points.Add(globalNbrij);
            todoQueue.Enqueue(nbrIj);
        }
    }
}
```

Представленный код находит соседние вершины по отношению к данной и проверяет их на наличие в списке точек касания, объединяя их в одно пятно размером максимум в 5 точек и присваивает уникальный `id` для текущего пятна контакта.

Далее необходимо посчитать периметр и площадь для текущего пятна контакта. В качестве алгоритма, реализующего это используется алгоритм Джарвиса. В листинге 1 приложения Б представлена реализация функции `ComputeJarvis`, вычисляющая выпуклую область контакта на основе алгоритма Джарвиса.

Размерным параметром пятна контакта является `oob`, которой требуется рассчитать для данной компоненты связи. В случае, если точек связи меньше 4, периметр, площадь и `oob` считаются как для примитивных геометрических объектов, в ином же случае используется функция `ComputeJarvis`. Блок кода, рассчитывающий данные параметры приведен в листинге 2 приложения Б.

После этого структура для пятна контакта полностью заполнена и можно переходить к физическим расчетам.

3.1.2. Вычисление физических параметров точек касания

На основе формул, приведенных в главе 2, можно осуществить вычисление нормальной и тангенциальных сил, а также пластической деформации в точке для определения момента разрушения почвы. Блок кода, в котором происходит итерирование по всем лучам, задевших объекты, и расчет необходимых сил приведен в листинге 4.

Листинг 4 – Расчет сил, действующих от почвы на объект

```
foreach (var hit in hitPositions)
{
   Vector2 ij = hit.Key;
   NodeRecord nr = m_grid_map[ij];
   float ca = nr.normal.y;
   nr.hit_level = hit.Value.abs_point.y;
   float p_hit_offset = ca * (nr.level_initial - nr.hit_level);
   nr.sigma = elastic_K * (p_hit_offset - nr.sinkage_plastic);
   Vector3 objectSpeed = terrain.GetSphereSpeed(hit.Value.abs_point);
   modifiedNodes.Add(ij);
   Vector3 N = nr.normal;
   float Vn =  Vector3.Dot(objectSpeed, N);
   Vector3 T = -(objectSpeed - (Vn * N));
   T = Vector3.Normalize(T);
   nr.sinkage = p_hit_offset;
   nr.level = nr.hit_level;
   nr.kshear += Vector3.Dot(objectSpeed, -T) * Time.deltaTime;
   if (nr.sigma > nr.sigma_yield)
   {
      nr.sigma = (Bekker_Kc * contactPatch.oob + Bekker_Kphi)
*Math.Pow(nr.sinkage, Bekker_n);
nr.sigma_yield = nr.sigma;
      double old_sinkage_plastic = nr.sinkage_plastic;
      nr.sinkage_plastic = nr.sinkage - nr.sigma / elastic_K;
      nr.step_plastic_flow = (nr.sinkage_plastic - old_sinkage_plastic) /
Time.fixedDeltaTime;
   }
   nr.sigma += -Vn * damping_R;
   double tau_max = Mohr_cohesion + nr.sigma * Mohr_mu;
   nr.tau = tau_max * (1.0 - Math.Exp(-(nr.kshear / Janosi_shear)));
   Vector3 Fn = N * areaCell * (float)nr.sigma;
   Vector3 Ft = T * areaCell * (float)nr.tau;
   Vector3 resaltF = Fn + Ft;
   hit.Value.contactable.AddForceAtPosition(resaltF, hit.Value.abs_point,
ForceMode.Force);
}
```

Для хранения значений используется список объектов структуры `NodeRecord`, в полях которой хранятся физические величины, необходимые для вычислений на следующих итерациях. Чтобы применить силы к объекту, используется встроенная в Unity функция `AddForceAtPosition`, которая применяет постоянную силу к объекту в заданной точке. Имея точку удара

луча с объектом, можно легко применить в ней силу к объекту. Также на данном шаге симуляции происходит смещение вершины Terrain, находящейся в каждой точке контакта. Смещение высоты происходит за счет стандартных функций объекта Terrain в Unity, а высота смещения рассчитывается как разница между текущей высотой вершины и высотой точки касания.

Для того чтобы объект мог дальше перемещаться по плоскости, используя стандартную физику движка, для него на сцене создается дополнительный объект, который переводится на слой Ignore Raycast – стандартный слой в Unity, который игнорируют лучи. В результате, добавочный элемент не обрабатывается поверхностью, как физический объект.

Таким образом, данный расчет происходит на каждом шаге симуляции с помощью функции `FixedUpdate`, которую движок вызывает каждый раз через заданный в шаге промежуток времени.

3.1.3. Деформирование поверхности

Возможность деформировать поверхность осуществляется в отдельном от расчетов классе, который отвечает за управление самим объектом Terrain – сетчатой поверхностью. В Unity есть удобный метод для изменения высот ландшафта `SetHeights`, которому необходимо передать массив точек высот текущей плоскости. Для выполнений процедуры корректного обновления высот реализован метод `Set`, который принимает на вход значения x и z текущей точки, а также значение `val`, на которое требуется изменить высоту. Код этого метода представлен в листинге 5.

Листинг 5 – Изменение высот деформируемой поверхности

```
public void Set(int x, int z, float val)
{
   x = (x + heightmap_width) % heightmap_width;
   z = (z + heightmap_height) % heightmap_height;
   heightmap_data[z, x] = val / terrain_data.heightmapScale.y;
}
```

Значения x и z берутся из координат текущего луча, задевшего объект, а в `val` передается значение высоты точки удара этого луча в объект, то есть

высоты погружения объекта в почву. Для применения этих изменения используется метод `Save`, который, используя описанную раннее функцию `SetHeights`, изменяет массив точек высот. Метод `Save` вызывается в встроенным в Unity методе `FixedUpdate`, который автоматически вызывается каждый шаг симуляции.

После реализованных механик объект Terrain на сцене обладает свойствами подвергаться деформациям на основе физических параметров почвы, что позволяет имитировать процесс уплотнения грунта после взаимодействия с объектом. Это достигается за счет использования физической модели Беккера, которая учитывают различные параметры почвы. В результате, при взаимодействии танковой модели с поверхностью, происходит реалистичное изменение структуры почвы, создавая эффект следов от гусениц.

На рисунке 6 представлен скриншот, демонстрирующий область почвы, подвергшуюся уплотнению после прохождения танка. Видно, что после движения танковой модели по поверхности, грунт был заметно изменен: на нем образовались характерные следы от гусениц.

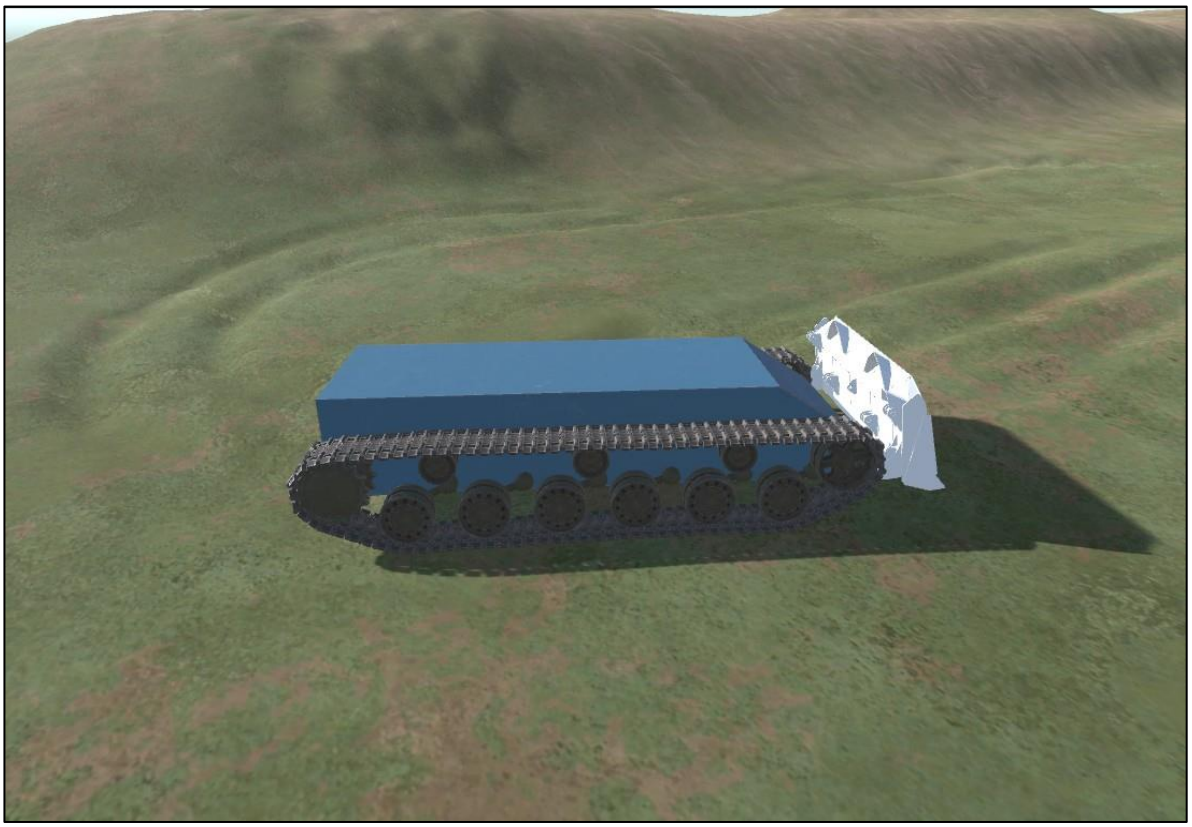

Рисунок 6 – Следы от гусениц танка на почве

3.2. Реализация модели создания гранул

3.2.1. Создание гранул

Для того, чтобы хранить и передавать рассчитанные раннее свойства гранулы в скрипте расчетов для деформируемой поверхности используется структура `GranulInfo`, хранящая в себе два поля: объем гранулы и точка для ее создания в мировых координатах сцены. Код, осуществляющий заполнение данной структуры находится в условии разрушения почвы, он приведен в листинге 6.

Листинг 6 – Расчет объема и позиции создаваемой гранулы

```
if (threhold >= flowFactor && (objectSpeed.x>0.1f || objectSpeed.z>0.1f))
{
   Vector2 direct_vec = TransformVector(objectSpeed.normalized);
   Vector3 creationDirection = (objectSpeed + nr.normal).normalized*ar-
eaCell;
   float curr_height = allRays[ij];
   float _next_height = curr_height;
   Vector2 ii = ij + direct_vec;
   allRays.TryGetValue(ii * 2, out _next_height);
   float vol = nr.sinkage * areaCell/3;
   if (curr_height > _next_height) _next_height = curr_height;
   Vector3 point = new Vector3(hit.Value.abs_point.x+ creationDirection.x,
_next_height + 4*vol, hit.Value.abs_point.z+creationDirection.z);
   if ((hit.Value.abs_point - point).magnitude <= vol) vol = 0;
   GranulInfo ginf = new GranulInfo(vol, point,ii);
   granuls[ij] = ginf;
}
```

В данном коде после сравнения рассчитанного раннее значения `threshold` с константным значением коэффициента разрушения `flowFactor` происходит расчет позиции для создания гранулы в направлении скорости объекта с учетом следующего луча на сетке вершин в этом направлении. Для хранения значений всех лучей имеется словарь `allRays`, где в роли ключа выступает структура `HitRecord`, а значением является вектор координат высоты поверхности, в которой инициализируется луч. Объем гранулы вычисляется как объем пирамиды, которую образует одна компонента связи пяти вершин, где высота – это разница между высотой погружения объекта и текущей высотой точки на сетке, а площадь основания – это площадь одной клетки поверхности.

Объект `GranulInfo` после инициализации передается в переопределенный метод Set, обновляющий высоты поверхности, который вызывает метод `CreateGranul`, отвечающий за создание гранулы на сцене. Код метода `CreateGranul` приведен в листинге 7.

Листинг 7 – Метод создания гранул на сцене

```
public void CreateGranul(GranulInfo ginf)
{
   if (ginf.volume == 0) return;
   float radius = Mathf.Pow((3f * ginf.volume) / (4f * Mathf.PI), 1f / 3f);
   if (radius > 0.1f)
   {
      int remainder = (int)(radius / 0.1f);
      for (int i = 0; i < remainder; i++)
      {
         Vector3 new_point = ginf.inst_point + 0.1f * i * Vector3.up;
         GameObject _granul = granulPool.GetGranule();
         _granul.transform.position = new_point;
         _granul.transform.localScale = new Vector3(0.2f, 0.2f, 0.2f);
         var granul_cont = _granul.GetComponent<GranulCont>();
         granul.GetComponent<Rigidbody>().mass = (float)PhysicalFoot-
printSphere.granDensity*ginf.volume/remainder;
         granul_cont.radius = 0.1f;
         granul_cont.my_obj = mySphere;
      }
   }
   else
   {
      GameObject _granul = granulPool.GetGranule(); //Instantiate(granul,
new_point, Quaternion.identity);
      _granul.transform.position = ginf.inst_point;
      _granul.transform.localScale = new Vector3(radius * 2f, radius * 2f,
radius * 2f);
      var granul_cont = _granul.GetComponent<GranulCont>();
      granul.GetComponent<Rigidbody>().mass = (float)PhysicalFoot-
printSphere.granDensity * ginf.volume;
      granul_cont.radius = radius;
      granul_cont.my_obj = mySphere;
   }
}
```

В данном методе определен фиксированный радиус гранулы равный 0,1. Радиус изначально рассчитанной гранулы вычисляется исходя из формулы объема сферы и делится на фиксированный. Таким образом получается необходимое количество гранул заданного размера, которые создаются методом `GetGranule` вспомогательного класса `GranulePool`. Вспомогательный класс представляет из себя управляющий гранулами пул объектов, который создает фиксированное количество гранул на сцене стандартным

методом `Instantiate`, а затем выключает их. Создание гранулы происходит с помощью включения созданных заранее гранул. Количество создаваемых изначально гранул определяется самим пользователем. Такая механика позволяет увеличить производительность за счет отсутствия необходимости использовать ресурсоемкий метод создания гранул на каждом шаге. В случае, если радиус изначальной гранулы меньше фиксированного, она создается в первоначальном виде. Помимо этого, каждая гранула является префабом (предварительно инициализированным объектом), на котором находится скрипт `GranulCont`, контролирующий физическое взаимодействие гранул.

3.2.2. Взаимодействие гранул

Для того, чтобы реализовать применение силы трения и изменение скоростей гранул на основе закона сохранения импульса, используется стандартная логика определения контактов движка. В Unity имеется встроенный метод `OnCollisionEnter`, который вызывается каждый раз, когда происходит касание объектов, и метод `OnCollisionStay`, который вызывается каждый шаг, когда объекты находятся в процессе касания. В методе `OnCollisionStay` происходит расчет и применение силы трения на основе информации точки контакта, которую хранит вызываемый метод. В листинге 8 представлена реализация расчета и применения силы трения в методе `OnCollisionStay`.

Листинг 8 – Расчет силы трения гранулы

```
private void OnCollisionStay(Collision collision)
{
    if (rb == null || collision.collider.attachedRigidbody == null) return;
        Vector3 normalForce = collision.contacts[0].normal * collision.im-
pulse.magnitude;
        float frictionForce = mu * normalForce.magnitude;
        rb.AddForce(-normalForce.normalized * frictionForce, Force-
Mode.Force);
}
```

В методе `OnCollisionEnter` происходит расчет и изменение скоростей гранул после их упругого столкновения. В листинге 9 представлена реализация этих расчетов.

Листинг 9 – Расчет скоростей гранул

```
private void OnCollisionEnter(Collision collision)
{
   if (rb == null) return;
   if (collision.transform.tag == "Granul")
   {
      Rigidbody otherRb = collision.rigidbody;
      Vector3 relativeVelocity = otherRb.velocity - rb.velocity;
      Vector3 newVelocity = rb.velocity + (1 + mu_coeff) * otherRb.mass /
(rb.mass + otherRb.mass) * relativeVelocity;
      rb.velocity = newVelocity;
      Vector3 otherNewVelocity = otherRb.velocity - mu_coeff * rb.mass /
(rb.mass + otherRb.mass) * relativeVelocity;
      otherRb.velocity = otherNewVelocity;
   }
}
```

В обоих методах используется информация о точке столкновения, хранящаяся в переменной `collision`, из которой можно получить текущие скорости объектов, нормаль в точке касания, а также их массы. Применение сил и изменение скоростей происходит через класс твердого тела гранулы.

3.2.3. Удаление гранул

После того как гранулы созданы, необходимо определить момент их удаления и возвращения в поверхность. Условием для этого является минимальная скорость и определенная заранее удаленность от взаимодействующего с ними объекта. То есть, когда гранулы находятся в определенном радиусе объекта и их скорость выше минимальной – они продолжают существовать на сцене. За удаление и возвращение гранул в поверхность отвечает абстрактный класс `GranulManager`, внутри которого осуществлена подписка на событие `SetGranuled`. Событие вызывается после условия удаления внутри класса `GranulCont`, привязанного к каждой грануле. Далее `GranulManager` вызывает все функции, подписанные на событие. В классе, отвечающем за управление террейном на данное событие подписана функция `SetGranul`, код которой приведен в листинге 10.

Листинг 10 – Функция возвращения гранул в поверхность

```
public void SetGranul(Vector3 pos, float radius)
{
   pos = World2Grid(pos);
   Vector2 ij = new Vector2(pos.x, pos.z);
   float Y = Get(ij) + radius / 5;
   Set(ij.x, ij.y, Y);
   for (int i = 0; i < 4; i++)
   {
      Vector2 ii = new Vector2(ij.x, ij.y) + neighbors4[i];
      float Y1 = Get(ii);
      if (Y>=Y1) Set(ii.x, ii.y, Y1 + radius /5);
      else Set(ij.x, ij.y, Y);
   }
}
```

В данной функции происходит следующее:

1) позиция гранулы, передаваемая в метод, переводится в локальные координаты поверхности;

2) определяется высота, на которую необходимо поднять точку поверхности, как 1/5 величины радиуса;

3) вокруг текущей координаты поверхности находятся еще 4 соседние вершины;

4) если высота первой вершины больше соседней, обновляем высоту соседней вершины, в противном случае обновляем первоначальную.

Таким образом, получается эффект сглаживания поднятой вершины. Место, в котором была удалена гранула поднимается на высоту 1/5 радиуса и шириной в 5 вершин.

Также можно изменить модель гранул со сферической на произвольную так, чтобы их объем был пропорционален. На рисунке 7 представлен скриншот сцены, на котором с помощью модели танка с отвалом было произведено разрушение материала почвы.

В данном примере обычные гранулы были заменены на модели камней с текстурами, объем которых почти равен объему соответствующих сфер. Данная погрешность допустима, так как влияет только на визуальную составляющую.

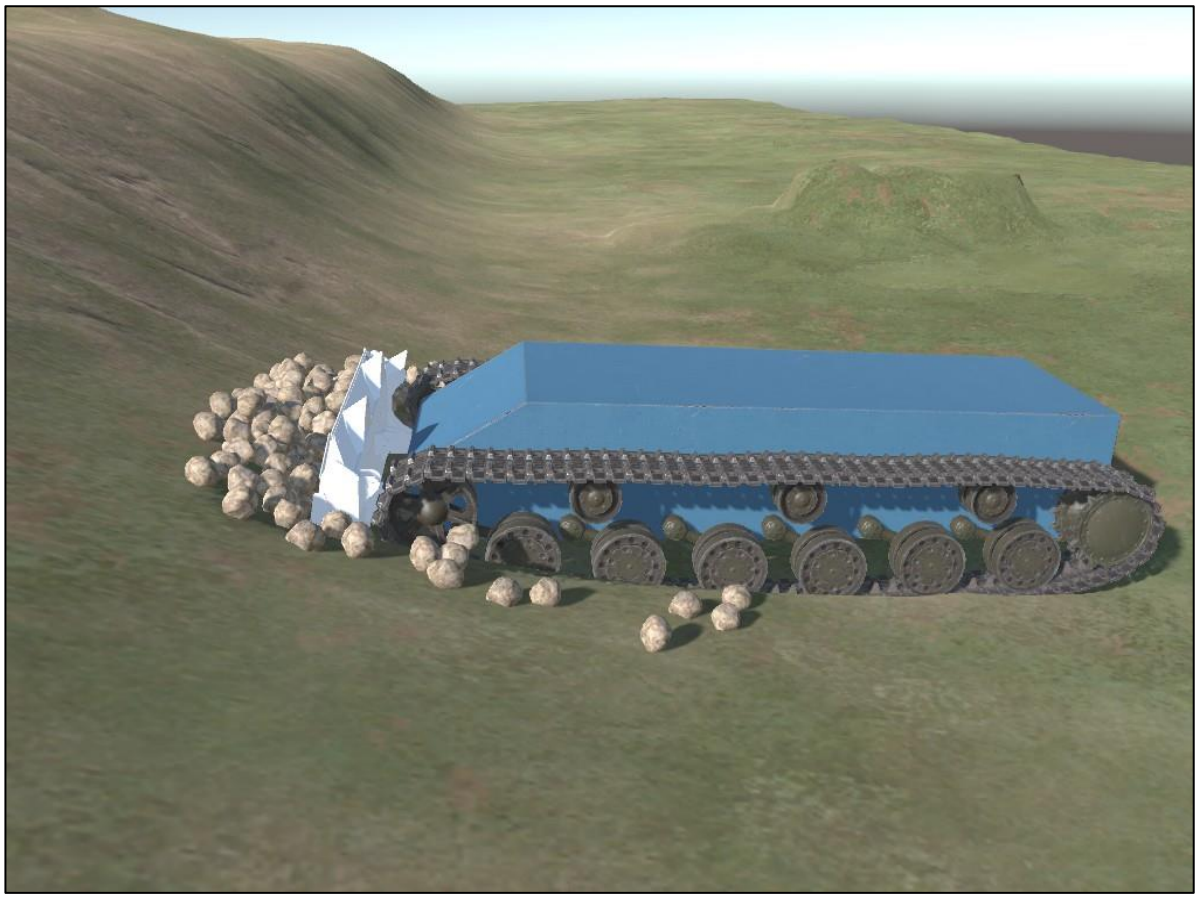

Рисунок 7 – Гранулы земли, выкопанные из почвы

3.3. Расчет и визуализация нагрузок

3.3.1. Расчет нагрузок

Для получения полной величины нагрузки на объект используется вычисленная по модели Беккера сила, действующая на объект, а также сила, с которой действуют созданные гранулы, то есть их вес с учетом нормали к объекту. Общая нагрузка на объект – это сумма сил от почвы к объекту на каждом луче и силы тяжести от каждой гранулы, соприкоснувшейся с объектом. Сила, действующая от почвы уже вычислена ранее, а для расчета сил тяжести от гранул используется третий закон Ньютона. В методе `OnCollisionEnter` класса `GranulCont` происходит вызов метода `CalculateGravitationalForce`, который реализует вычисление силы тяготения на основе информации момента касания. Код метода `CalculateGravitationalForce` приведен в листинге 11.

Листинг 11 – Вычисление силы, действующей от гранулы к объекту

```
Vector3 CalculateGravitationalForce(GameObject obj1, float m1, GameObject
obj2, float m2)
{
   Vector3 direction = obj2.transform.position - obj1.transform.position;
   float distance = direction.magnitude;
   direction.Normalize();
   float forceMagnitude = G * (m1 * m2) / Mathf.Pow(distance, 2);
   Vector3 force = forceMagnitude * direction;
   return force;
}
```

В классе, отвечающем за расчет сил, имеется статическое поле в виде словаря, хранящая значения векторов сил по ключу объекта. Вычисленные силы почвы в цикле, проходящим по всем лучам, сразу записываются в словарь, а силы, действующие от гранул, записываются в момент касания, используя статическое поле класса. Итоговая нагрузка для каждого объекта равна сумме вычисленных сил и выражается в Ньютонах [Н]. На каждой итерации словарь очищается для обновления значения нагрузок.

3.3.2. Визуализация нагрузок

Для того, чтобы пользователь мог следить за изменением нагрузок, словарь со списком сил и объектов, к которым они применены, выводится в UI панель, закрепленную на экране. Статическое поле словаря используется в классе `ForceDisplay`, который прикреплен к UI элементу Canvas [16], отображающему панель с нагрузками. У объекта из словаря берется текстовое значение его имени и выводится со значением нагрузки, выраженной в Ньютонах. Таким образом, на панели отображается нагрузка, действующая на каждый объект на сцене на данном шаге симуляции. На рисунке 8 представлен скриншот, на котором изображена UI панель со списком нагрузок на каждый объект в процессе симуляции.

Рисунок 8 – Скриншот панели отображения нагрузок

На данном скриншоте представлены результаты отображения нагрузок в ходе симуляции. Вывод имеет следующий формат записи:

1) имя объекта;
2) двоеточие;
3) нагрузка, выраженная в Ньютонах.

Также для того, чтобы пользователь мог собирать и оценивать данные, значения нагрузок сериализуются в `json` файл в папке с проектом.

Вывод по третьей главе

В соответствии со спроектированной архитектурой модели была разработана модель деформируемой поверхности почвы, основанная на физических параметрах, влияющих на ее поведение. Также был реализован гранулированный подход, позволяющий имитировать разрушение почвы и превращение ее в гранулы.

4. ТЕСТИРОВАНИЕ

В ходе тестирования проверялось соответствие поведения почвы на изменения ее параметров согласно модели Беккера. Влияние параметров определялось путем варьирования их значений для имитирования одного из семи видов почвы. Значения каждого из параметров представлены в таблице 2.

Таблица 2 – Типы почвы с соответствующими им параметрами

Тип почвы	**Bekker_Kphi**	**Bekker_Kc**	**Bekker_n**	**Mohr_cohesion**	**Mohr_mu**	**Janosi_sh–ear**	**Elastic_K**	**damping_R**
Песок	15000	2000	1,6	100	30	0,02	250000	50
Глина	300000	50000	1,4	350	40	0,1	600000	200
Гравий	400000	20000	1,1	20	35	0,5	1000000	100
Торф	50000	5000	1,6	50	10	0,01	200000	20
Супесь	250000	40000	1,2	30	25	0,03	500000	80
Чернозем	300000	30000	1,2	300	30	0,05	1000000	100
Твердый грунт	2000000	40000	0,9	300	30	0,05	50000000	100

На основании этих данных был проведен ряд тестов, в которых проверялось, соответствует ли поведение имитируемой модели с реальным типом почвы на основе ее параметров. Результаты тестирования для каждого из спроектированных типов почвы приведены в таблице 3.

Таблица 3 – Результаты тестирования модели почвы

№	**Название теста**	**Изменение в модели**	**Ожидаемый результат**	**Тест пройден?**
1.	Проверка модели песка	Настройка параметров почвы, соответствующих песку	Почва легко уплотняется, легко разрушается, увеличивается степень пробуксовки техники, выкапываемая почва имеет рассыпчатый вид	Да
2.	Проверка модели глины	Настройка параметров почвы, соответствующих глине	Почва вязкая, имеет высокое сцепление и сопротивление, техника может застревать, выкапываемая почва липкая и плотная	Да

Окончание таблицы 3

№	Название теста	Изменение в модели	Ожидаемый результат	Тест пройден?
3.	Проверка модели гравия	Настройка параметров почвы, соответствующих гравию	Почва твердая, легко удерживает технику, минимальная пробуксовка, выкапываемая почва имеет редкую структуру	Да
4.	Проверка модели торфа	Настройка параметров почвы, соответствующих торфу	Почва очень мягкая, сильно деформируется, техника легко застревает, выкапываемая почва имеет рассыпчатый вид	Да
5.	Проверка модели супеси	Настройка параметров почвы, соответствующих песку	Почва умеренно мягкая, техника умеренно пробуксовывает, легко разрушается, выкапываемая почва имеет рассыпчатый вид	Да
6.	Проверка модели чернозема	Настройка параметров почвы, соответствующих песку	Почва умеренно мягкая, хорошо удерживает технику, минимальная пробуксовка, выкапываемая почва имеет рыхлую структуру	Да
7.	Проверка модели твердого грунта	Настройка параметров почвы, соответствующих песку	Почва очень твердая, почти не деформируется, неровности почвы вызывают затруднения в движении техники, тяжело разрушается, выкапываемая почва имеет редкую структуру	Да

Вывод по четвертой главе

В четвертой главе были приведены результаты тестирования модели на основе изменения ее параметров. Сравнение результатов тестов показало соответствие с предполагаемыми результатами.

ЗАКЛЮЧЕНИЕ

В рамках данной работы была разработана модель почвы с возможностью рассчитывать нагрузки и имитировать строительные процессы. Разработанная модель имеет физические параметры, которые пользователь может настраивать для управления типом почвы, определяющим ее поведение в симуляции. Кроме того, модель была реализована с возможностью настраивать точность и ее поведение на основе оптимизируемых параметров. Модель почвы была разработана в физическом движке Unity и представляет из себя инструмент для внедрения в проекты физических симуляций.

В ходе данной работы были решены следующие задачи.

1. Произведен обзор литературы и существующих решений по предметной области.
2. Спроектирована и реализована модель деформируемой поверхности в физическом движке.
3. Спроектирована и реализована гранулированная модель почвы для имитации разрушения почвы в физическом движке.
4. Реализован расчет и вывод нагрузок на взаимодействующие с почвой тела.
5. Проведено тестирование модели на основе изменения различных параметров почвы.

ЛИТЕРАТУРА

1. Role of Computer Simulation Technology in Modern Engineering. [Электронный ресурс] URL: https://www.simscale.com/blog/role-simulation-technology-engineering (дата обращения: 19.04.2024 г.).

2. Документация физического движка Chrono. [Электронный ресурс] URL: https://api.projectchrono.org (дата обращения: 10.02.2024 г.).

3. Документация игрового движка Unreal Engine 5.3. [Электронный ресурс] URL: https://docs.unrealengine.com/5.3/en-US (дата обращения: 10.02.2024 г.).

4. Документация игрового движка Unity. [Электронный ресурс] URL: https://docs.unity3d.com/Manual (дата обращения: 10.02.2024 г.).

5. Документация физического движка Godot. [Электронный ресурс] URL: https://docs.godotengine.org/en/stable (дата обращения: 10.02.2024 г.).

6. Игровой движок Havok. [Электронный ресурс] URL: https://www.havok.com/havok-physics (дата обращения: 10.02.2024 г.).

7. Игровой движок Cry Engine. [Электронный ресурс] URL: https://docs.cryengine.com (дата обращения: 24.03.2024 г.).

8. Discrete element method. [Электронный ресурс] URL: https://docs.software.vt.edu/abaqusv2022/English/SIMACAEANLRef-Map/simaanl-c-demanalysis.htm (дата обращения: 24.03.2024 г.).

9. Инструкция по использованию Mesh в Unity. [Электронный ресурс] URL: https://docs.unity3d.com/ScriptReference/Mesh.html (дата обращения: 21.02.2024 г.).

10. Инструкция по использованию Terrain в Unity. [Электронный ресурс] URL: https://docs.unity3d.com/ru/530/Manual/script-Terrain.html (дата обращения: 21.02.2024 г.).

11. Документация к использованию деформируемых Terrain в движке Chrono. [Электронный ресурс] URL: https://api.projectchrono.org/vehicle_terrain.html (дата обращения: 22.02.2024 г.).

12. Инструкция по использованию RayCast в Unity. [Электронный ресурс] URL: https://docs.unity3d.com/ScriptReference/Physics.Raycast.html (дата обращения: 21.02.2024 г.).

13. Bekker's Terramechanics Model for Off-Road Vehicle Research. [Электронный ресурс] URL: https://www.researchgate.net/publication/235069513_Bekker's_Terramechanics_Model_for_Off-Road_Vehicle_Research (дата обращения: 26.02.2024 г.).

14. Расчеты грунтов по модели Кулона-Мора. [Электронный ресурс] URL: https://kb-sp.ru/informatsiya/geotehnichskaya_model_kulona_mora (дата обращения: 23.03.2024 г.).

15. A. Bernard-Champmartin, Olivier Poujade, Mathiaud Julien, J.-M. Ghidaglia. Modelling of an Homogeneous Equilibrium Mixture Model. // Acta Applicandae Mathematicae. – 2013. – Т. 129. – С. 1-21.

16. Инструкция по использованию Canvas в Unity. [Электронный ресурс] URL: https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/UICanvas.html (дата обращения: 23.03.2024 г.).

ПРИЛОЖЕНИЯ

Приложение А. Диаграмма деятельности работы модели почвы

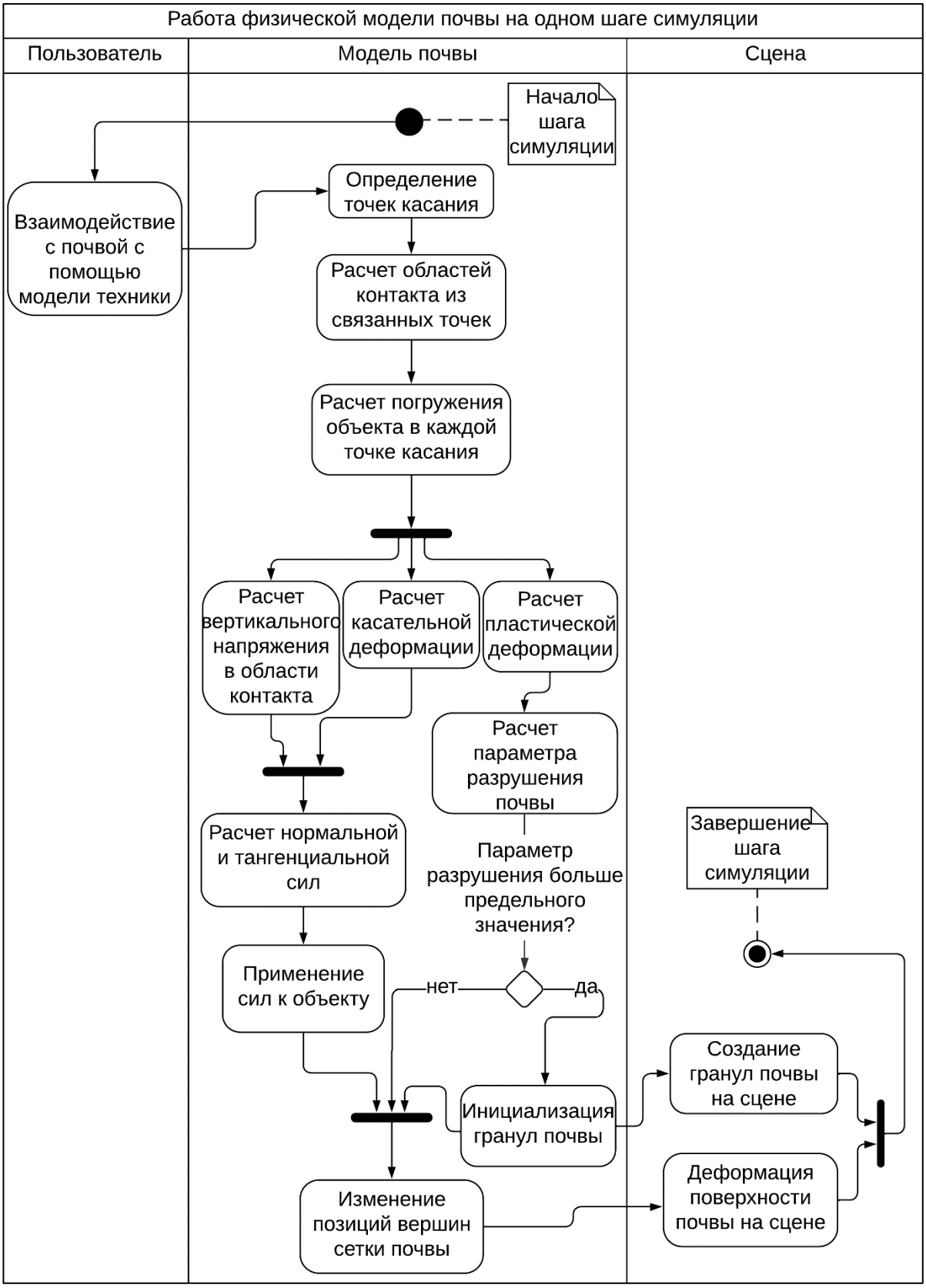

Рисунок 1 – Диаграмма деятельности работы модели почвы на одном шаге симуляции

Приложение Б. Реализация вычислений поверхности объекта

Листинг 1 – Функция для расчета области контакта объекта с почвой

```
public void ComputeJarvis(List<Vector2> points, int n, out double area, out
double perimeter)
    {
        bool[] added = Enumerable.Repeat(false, points.Count).ToArray();
        area = 0;
        perimeter = 0;
        int addCount = 0;
        int first = 0;
        for (int i = 1; i < points.Count; i++)
        {
            double dx = points[i].x - points[first].x;
            if (dx < 0)
                first = i;
            else if (dx == 0 && points[i].y < points[first].y)
                first = i;
        }
        addCount++;
        int crt = first;
        do
        {
            int next = 0;
            while (next == crt || added[next])
                next = (next + 1) % n;
            for (int i = 0; i < n; i++)
            {
               if (Orientation(points[crt], points[i], points[next]) == -1)
                {
                    next = i;
                }
            }
            for (int i = 0; i < n; i++)
            {
                if (i != crt && i != next && !added[i] && Orienta-
tion(points[crt], points[i], points[next]) == 0 &&
                 InBetween(points[crt], points[i], points[next]))
                {
                    added[i] = true;
                    addCount++;
                }
            }
            added[next] = true;
            addCount++;
            perimeter += (points[next] - points[crt]).magnitude;
            area += SignedArea(points[next], points[crt], Vector2.zero);

         return;
         }
            crt = next;
       } while (crt != first);
    }
```

Листинг 2 – Расчет размерного параметра пятна контакта

```
if (contactPatch.points.Count != 0)
        {
            List<Vector2> points = contactPatch.points;
            int n = points.Count;
```

```
            if (n > 0)
            {
                double area;
                double perimeter;
                switch (n)
                {
                    case 1:
                        contactPatch.perimeter = 0;
                        contactPatch.area = 0;
                        contactPatch.oob = 0;
                        break;
                    case 2:
                        perimeter = (points[1] - points[0]).magnitude;
                        contactPatch.perimeter = perimeter;
                        contactPatch.area = 0;
                        contactPatch.oob = 0;
                        break;
                    case 3:
                        area = 0.5 * Math.Abs(SignedArea(points[0],
points[1], points[2]));
                        perimeter = (points[1] - points[0]).magnitude +
                            (points[2] - points[1]).magnitude +
                            (points[0] - points[2]).magnitude;
                        contactPatch.perimeter = perimeter;
                        contactPatch.area = area;
                        contactPatch.oob = perimeter/ (2* area);
                        if (area == 0)
                        {
                            contactPatch.oob = 0;
                        }
                        break;
                    default:
                        ComputeJarvis(points, n, out area, out perimeter);
                        area *= 0.5;
                        contactPatch.perimeter = perimeter;
                        contactPatch.area = area;
                        contactPatch.oob = perimeter / (2 * area);

                        if (area == 0)
                        {
                            contactPatch.oob = 0;
                        }
                        break;
                }
            }
        }
```