Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
(ФГБОУ ВО «АмГУ»)

Факультет математики и информатики
Кафедра информационных и управляющих систем
Направление подготовки 09.04.04 Программная инженерия
Направленность (профиль) образовательной программы Управление разработкой программного обеспечения

ДОПУСТИТЬ К ЗАЩИТЕ
Зав. кафедрой
_____ А.В. Бушманов
«_____»_____ 2023 г.

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

на тему: Разработка мобильной игры в среде разработки Unity

Исполнитель
студент группы 157-ом _____ М.А. Свитецкий
(подпись, дата)

Руководитель
профессор, доктор техн. наук _____ А. Д. Плутенко
(подпись, дата)

Руководитель научного
содержания программы
магистратуры
профессор, доктор техн. наук _____ И.Е. Еремин
(подпись, дата)

Нормоконтроль
доцент, канд.техн. наук _____ Л.В. Никифорова
(подпись, дата)

Рецензент _____ С.В. Щербаков
(подпись, дата)

Благовещенск, 2023

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательная учреждение высшего образования
АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
(ФГБОУ ВО «АмГУ»)

Факультет математики и информатики
Кафедра информационных и управляющих систем

УТВЕРЖДАЮ
Зав. кафедрой
_____ А.В. Бушманов
подпись
«_____»_____2023 г.

З А Д А Н И Е

К магистерской диссертации студента группы 157-ом

Свитецкого Максима Андреевича

1. Тема магистерской диссертации: Разработка мобильной игры в среде разработки Unity

(Утверждено приказом от 21.02.2023 № 442-уч)

2. Срок сдачи студентом законченной работы (проекта) 20.06.2023

3. Исходные данные к магистерской диссертации: предметная область, отчёты по практической подготовке, результаты выступления на научных конференциях, учебная литература, интернет-ресурсы.

4. Содержание магистерской диссертации (перечень подлежащих разработке вопросов): анализ предметной области проводимого исследования, проектирование программного продукта; реализация и тестирование программного продукта, практическая значимость полученных результатов.

5. Рецензент магистерской диссертации:

6. Дата выдачи задания 30.01.2023

Руководитель выпускной квалификационной работы: Плутенко А.Д., профессор, доктор техн. наук

Задание принял к исполнению (30.01.2023) _____

РЕФЕРАТ

Магистерская диссертация содержит 81 с., 35 рисунков, 1 таблицу, 53 источника

МОБИЛЬНЫЕ ВИДЕОИГРЫ, UNITY, СИМУЛЯЦИЯ ВОДЫ, МЕТОД СГЛАЖЕННЫХ ЧАСТИЦ, SPH

Объектом исследования магистерской работы является симуляция воды для видеоигр на мобильном устройстве.

Предметом исследования являются способы моделирования жидкостей в видеоиграх, возможность их применения на мобильных устройствах.

Целью работы является создание системы для моделирования движения несжимаемого потока жидкости в среде разработки Unity

Для достижения цели выполнены следующие задачи:

- проведен анализ предметной области;
- выполнен обзор существующих решений;
- описана математическая модель и алгоритм расчетов;
- выполнена разработка программного продукта;
- проведено тестирование разработанного программного продукта.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Современные игры стремятся предоставить игроку максимально реалистичный опыт, поэтому большое внимание уделяется точности и визуальной достоверности физических процессов. В частности, детализированная и реалистичная симуляция жидкости стала неотъемлемой частью современной компьютерной графики. Разработано множество алгоритмов для моделирования и визуализации движения жидкости. Однако расчеты до сих пор требуют достаточно высоких вычислительных мощностей. При этом вода – одна из самых сложных и ресурсоемких составляющих в видеоиграх. Поэтому часто она представлена только визуально, и многие ее физические свойства, которые могли бы использоваться в игровых механиках, теряются.

Но в последние годы производительность мобильных устройств возросла достаточно, чтобы добиться визуально достоверной симуляции поведения разных типов жидкости при достаточной частоте кадров. Естественно, их вычислительная мощность все еще не соответствует обычным настольным компьютерам, поскольку они должны работать от ограниченного источника питания и имеют большие ограничения по размеру. Это накладывает определенные ограничения на использование существующих решений для симуляции жидкости в видеоиграх поскольку они, как правило, ориентированы на мощные дискретные графические процессоры.

К тому же у многих популярных сред разработки игр отсутствуют встроенные инструменты для симуляции жидкостей, а сторонние плагины в большинстве представляют собой набор графических шейдеров, позволяющий визуализировать поверхность жидкости, но не способные симулировать физику её движения.

Поэтому было решено создать собственную систему симуляции жидкости для среды разработки Unity, которая позволила бы разработчику представить жидкость в виде интерактивного объекта, способного взаимодействует с другими твердыми телами на сцене.

Для этого было выделено несколько подзадач:

– требуется проанализировать существующие подходы к моделированию жидкостей в реальном времени и выбрать самый подходящий;

– используя выбранный метод моделирования описать математическую модель и построить алгоритм расчетов;

– создать программную реализацию построенного алгоритма. Продумать возможности оптимизации расчетов за счет использования встроенных инструментов Unity и выполнения параллельных вычислений;

– разработка графических шейдеров для визуализации поверхности жидкости.

1 ОБЩАЯ ХАРАКТЕРИСТИКА ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Программные средства симуляции физических процессов

Создание видеоигр является довольно трудоемким процессом. В случае крупных проектов работы могут вестись годы, даже при наличии большой и опытной команды разработчиков. Современные игровые движки позволяют упростить этот процесс, предоставляя разработчикам множество полезных инструментов для работы.

Согласно определению данному в [1], игровой движок – это главный программный продукт, используемый при разработке интерактивных графических приложений, например видеоигр. Обычно он предоставляет инструменты для работы с двумерной и трехмерной графикой, звуком, физическими компонентами, сценариями поведения объектов, анимацией, созданием пользовательского интерфейса и прочими игровыми процессами. Также движки используются для сборки проекта под разные целевые платформы, например мобильные устройства, консоли, настольные компьютеры. Поэтому их использование значительно ускоряет процесс разработки.

В [2] отмечается, что одной из важнейших задач игрового движка является симуляция физических процессов. Взаимодействия объектов в сцене, когда к ним применяются внешние силы (например воздействие игрока или гравитация), должны быть физически корректными. Для обнаружения столкновений применяются коллайдеры – это специальные компоненты, представляющие упрощенную границу объекта. Для задания свойств объекта применяются физические материалы. Они позволяют настраивать такие параметры моделирования как масса, трение, упругость объекта. Учитывая примененные к объектам компоненты, игровой движок обрабатывает их перемещения и столкновения. При этом выделяют твердые тела, которые подчиняются воздействию внешних сил, и статичные объекты, которые не меняют свое положение. Стоит отметить, что для создания эффекта погружения у игрока, подобное моделирование должно быть достаточно точным.

Согласно определению данному в [3], физически движком называют программное средство для моделирования физических процессов и явлений в виртуальной сцене. Зачастую физические движки не являются самостоятельными программными продуктами, а входят в состав других программ.

В зависимости от специфики работы можно выделить игровые и научные физические движки:

Первый тип используется как компонент игрового движка. Так как видеоигры являются интерактивными приложениями, главной задачей физического движка является моделирование физических процессов в реальном времени. При этом обычно не требуется высокая точность вычислений, поэтому для повышения производительности часто используются упрощенные модели взаимодействия.

В то же время в научных физических движках важна именно высокая точность моделирования физических процессов, где скорость выполнения вычислений не является приоритетом.

Моделирование физических процессов является неотъемлемой частью современных видеоигр. Однако это требует от команды разработчиков больших трудозатрат и высокой квалификации. Поэтому многие игровые движки имеют встроенные физические движки.

Согласно [4], встроенный в Unity физический движок поддерживает работу с твердыми телами, тканями, процедурную анимацию типа Ragdoll. Также существуют настройки физических материалов, влияющих на процесс моделирования (например трение, упругость и т. д.). При этом физические компоненты предусмотрены для 2D и 3D проектов. Также доступно управление объектом напрямую из скрипта, что позволяет создавать логику симуляции в случае необходимости.

Поскольку большая часть инструментов по симуляции физики в Unity3d создана для симуляции взаимодействия твёрдых тел, при моделировании движения воды от использования внутренних инструментов среды приходится отказаться, а значит требуется реализовывать собственные механизмы расчётов симуляции движения несжимаемого потока жидкости.

1.2 Особенности оптимизации мобильных игр в среде Unity

Производительность является одной из основных проблем при симуляции жидкостей на мобильных устройствах. Поэтому важным аспектом при разработке является оптимизация. Среди подходов к повышению производительности мобильных приложений часто можно выделить:

Минимизация вызовов отрисовки. Для быстродействия приложения важна не только скорость отрисовки объекта, но и количество вызовов отрисовки, при которых используются ресурсоемкие операции передачи данных в память GPU, поэтому их число должно быть минимальным. С этой целью в приложении нужно использовать объединение мешей (общее использование одного и того же экземпляра материала при визуализации). Также желательно использовать «Атласы текстур» – это набор различных текстур, упакованных в специальный объект. В [5] отмечается, что данный прием позволит существенно упростить GPU доступ к текстурам.

Желательно также применять технологию Occlusion culling, которая указывает Unity визуализировать только те объекты, которые попадают в кадр, что позволяет уменьшить число вызовов отрисовки. В масштабных играх с открытым миром прирост производительности может составить до 30%.

Использование пакетной обработки вызовов отрисовки. Это метод оптимизации, который объединяет меши объектов, чтобы Unity могла отображать их за меньшее количество вызовов. Для этого предусмотрено 2 встроенных метода пакетной обработки:

Static batching используется для статичных объектов, таких как горы и здания. Unity объединяет их вместе и отрисовывает за один проход.

Dynamic batching позволяет группировать похожие вершины меша и отрисовывать их вместе. Стоит учесть, что такое преобразование выполняется на CPU.

Использование оптимальных размеров текстур, что зависит от аппаратных характеристик целевых устройств. Лучше ориентироваться на бюджетные устройства, что позволит охватить максимальную аудиторию. Также нужно

учесть, что слишком большие текстуры не могу быть обработаны графическим процессором, поэтому они упрощаются CPU во время запуска сцены. Это увеличивает время загрузки и ухудшает итоговое качество текстур. Так как объем кеша и видеопамяти в мобильных устройствах не так велик, лучше многократно использовать одну и ту же текстуру, вместо множества различных [6].

Также согласно [7] желательно использовать квадратные текстуры со стороной, кратной степени 2, так как в этом случае графический процессор лучше справляется со сжатием.

Еще одним способом оптимизации является использование в шейдерах типов переменных с минимально возможной точностью, так как мобильные графические процессоры очень чувствительны к точности вычислений. Также желательно отказаться от преобразования форматов.

Также стоит рассмотреть способы оптимизации файлов мешей:

Уменьшение количества полигонов у 3D моделей. Так как к компонентам SkinnedMeshRenderer нельзя применить пакетную обработку, это единственный способ оптимизации анимированных объектов. При этом уменьшение числа вершин можно компенсировать за счет применения специальных текстур и поверхностных шейдеров, поэтому большая часть пользователей не заметит снижения детализации объекта.

Сжатие меша встроенными средствами Unity. Для этого существует две настройки:

Vertex Compression — это параметр, который влияет на все файлы мешей в проекте. Это позволяет использовать типы данных более низкой точности для хранения данных меша. Это уменьшает объем занимаемой памяти, немного уменьшает размер файла и может повысить производительность графического процессора [8]. Потенциальным недостатком является потеря точности.

Mesh Compression применяется к каждому мешу отдельно. Эта настройка сжимает данные файл меша на диске. Потенциальными недостатками являются увеличение времени загрузки (так как при загрузке меша применяется алгоритм декомпрессии) и возможность появления артефактов.

Также можно воспользоваться сторонними программами для работы с трехмерной графикой и попытаться оптимизировать меш с помощью имеющихся в них инструментов.

Использование настройки Optimize Meshes позволит Unity переупорядочивать вершины меша чтобы улучшить попадание во внутренний кеш, что позволит увеличить скорость отрисовки. Эта настройка по умолчанию включена для всех импортируемых файлов 3D моделей, но для процедурно генерируемых мешей нужно вызывать метод Mesh.Optimize. Эта операция может занять несколько секунд или больше для сложных мешей, поэтому обычно она вызывается в момент загрузки сцены.

Используя приведенные выше методы оптимизации можно добиться значительного роста производительности приложения.

1.3 Описание существующих методов решения задачи

В общем случает движение жидкости описывается уравнениями Навье-Стокса (НС). Но подходы к решению данного набора уравнений могут отличаться, в зависимости от характера расчетов.

Например, методы гидродинамического моделирования, используемые в компьютерной графике, отличаются от своих аналогов в вычислительной гидродинамике (CFD) в технике. Моделирование CFD используется для получения точных численных результатов для проектирования конструкций и машин.

Напротив, моделирование жидкости для компьютерной графики преследует следующие три основные цели:

– визуальная достоверность. Численная точность играет гораздо меньшую роль, чем в CFD;

– эффективность: моделирование жидкостей требует больших вычислительных ресурсов, поскольку требует решения сложных уравнений, описывающих движение потока. В то время как в инженерных приложениях производительность может не быть ключевым вопросом, требования к производительности в приложениях компьютерной графики, как правило, более строгие. Интерактивные приложения требуют, как минимум 24 обновления в секунду;

– стабильность: в большинстве графических приложений время моделирования определяется пользователем произвольно, например в видеоиграх. Следовательно, для запуска моделирования в течение длительных периодов времени требуются стабильные алгоритмы (в симуляции не должно быть возрастающих со временем ошибок) [9].

Кроме того, часто для приложений в реальном времени требуется, чтобы моделирование не являлось физически точным результатом, а соответствовало определенному сценарию, для достижения визуального эффекта. Например, может потребоваться, чтобы поток жидкости принимал заданную пользователем форму.

Этим целям придается разное значение в зависимости от сценария приложения. Например, интерактивные приложения, такие как видеоигры, требуют высокоэффективных вычислений, поскольку они обычно работают на неспециализированном оборудовании. Высокая эффективность в этом случае часто достигается за счет потери визуального качества. В такого рода симуляциях численная устойчивость является ключевым фактором, поскольку время моделирования произвольно и обычно определяется пользователем. Другие приложения, такие как визуальные эффекты в киноиндустрии, требуют высокого уровня визуальной детализации и для них требования к производительности не такие строгие. Эти симуляции часто проводятся на специализированном оборудовании, которое может выполнять большое количество одновременных операций.

В целом, к моделированию жидкости можно выделить два основных подхода:

Как указано в [10] в подходе Эйлера (рисунок 1) жидкость характеризуется значением функций, описывающих изменение какой-либо величины (давления, скорости, плотности, импульса, температуры и др.), в фиксированных точках пространства. Этот метод позволяет определить мгновенные значения интересующей величины в фиксированных точках «контрольного объема». Совокупность мгновенных значений каждой из этих величин называют её полем. В общем случае поля различных величин могут меняться во времени и по координатам.

В методе Лагранжа движущийся объем жидкости делится на отдельные части, называемые «частицами». Считается, что движение объема жидкости определено, если для каждой частицы этого объема известен вектор, определяющий ее положение, как функцию от начального положения и времени.

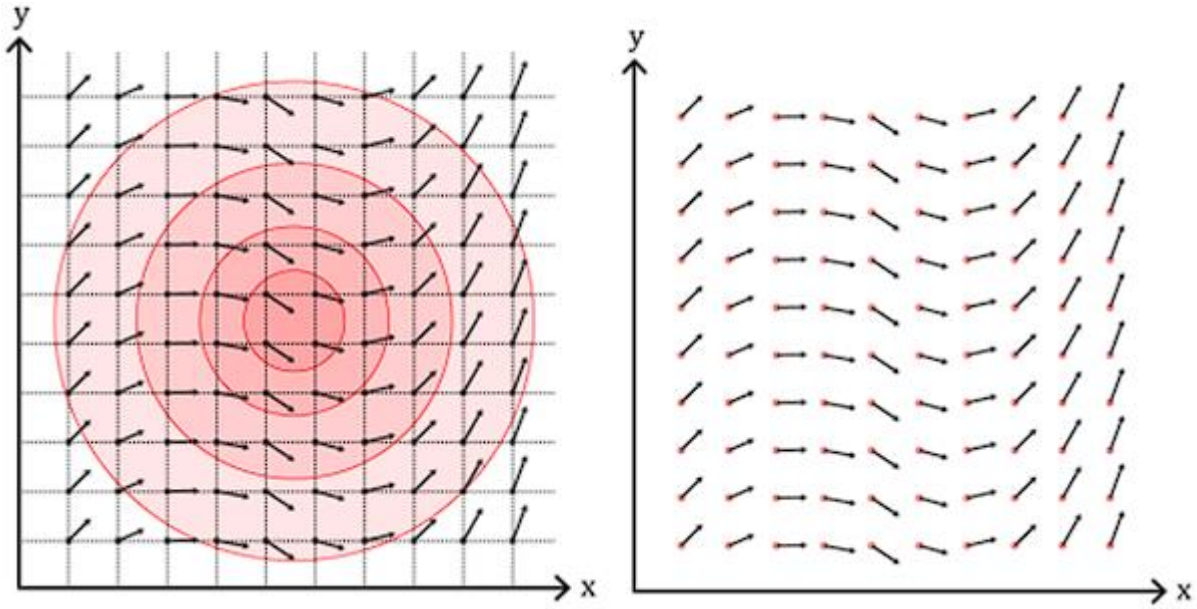


Рисунок 1 – Подход Эйлера (слева) и подход Лагранжа (справа)

На основе данных подходов существуют несколько методов моделирования. Наиболее популярными являются сеточные методы, основанные на подходе Эйлера, и методы сглаженных частиц (SPH), основанные на подходе Лагранжа.

Моделирование жидкости, основанное на подходе Эйлера, отслеживает свойства жидкости в фиксированных (дискретных) точках пространства. Эти свойства задаются либо скалярными, либо векторными полями, которые часто определяются в центре отдельных ячеек сетки. Симуляции на основе сетки бывают разных форм: фиксированная сетка, адаптируемая сетка и сетка с высокими ячейками (рисунок 2).

Простейшей формой является однородная фиксированная сетка. Здесь пространство разделено на ячейки, как правило, одинакового размера. Преимущество этой версии в том, что она позволяет выполнять быстрый поиск значений в ячейках, поскольку сетка может быть загружена в память при запуске моделирования. Недостатком является то, что это также расточительно, так как вероятно, что большое количество ячеек никогда не будет использовано. Кроме того, если

размер ячеек недостаточно мал, некоторые мелкие детали будут потеряны в областях с высокой активностью.

Адаптируемая сетка, как следует из названия, имеет неоднородную сетку, в которой области с низкой активностью будут представлены как большие ячейки, тогда как области с высокой активностью (например, область с завихренностью) будут предоставлены как ячейки меньшего размера. Хотя эта версия имеет очевидные преимущества по сравнению с фиксированной сеткой, сложно построить стабильную сетку с быстрым поиском.

Сетка с высокими ячейками подобно адаптируемой сетке сосредотачивает вычислительную мощность на областях с высокой активностью, которые обычно находятся вблизи поверхности и вблизи границ. Разница в том, что все ячейки ниже определенного расстояния до поверхности будут преобразованы в одну высокую ячейку в каждом столбце. Хотя эта версия имеет те же недостатки, что и адаптируемая сетка, большое преимущество заключается в том, что вычислительная мощность сосредоточена только на области вблизи поверхности, а все остальное игнорируется.

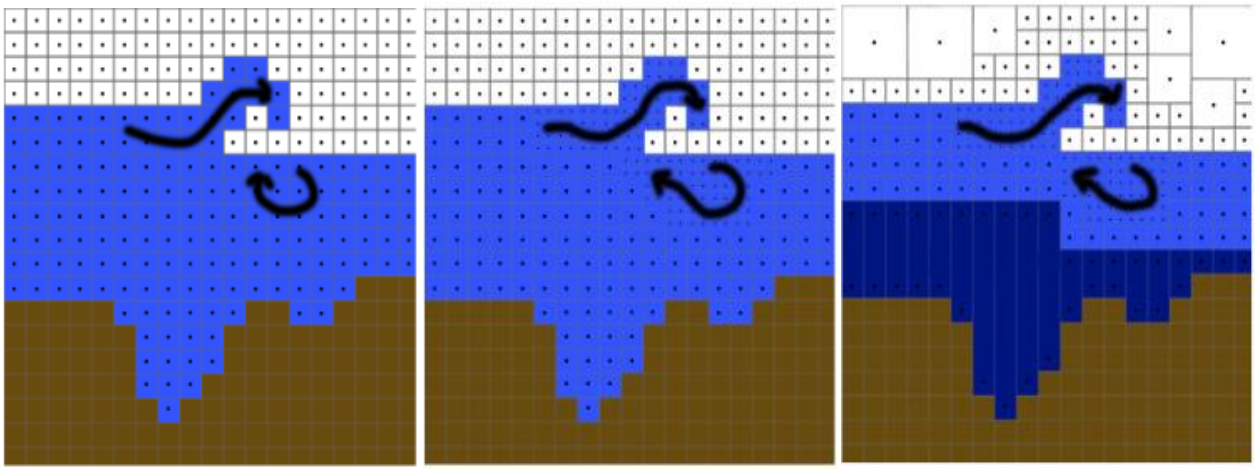


Рисунок 2 – Эйлеровы сетки: фиксированная (слева), адаптируемая сетка (по центру), сетка с высокими ячейками (справа)

Относительно сложная с точки зрения производительности часть эйлерова моделирования жидкости заключается в обеспечении сохранения несжимаемости жидкости. Один из способов сделать это — оценить, насколько меняется

количество жидкости в клетке и в каком направлении она течет.

Лагранжевы методы, напротив, не требуют явной дискретизации всей области. Дискретные узлы моделирования, например частицы, управляют динамическим поведением жидкостей. Методы лагранжевых частиц не зависят от ограничивающей рамки, определяющей область моделирования. Кроме того, можно представить жидкость с произвольным уровнем детализации, контролируя количество частиц, определяющих движение потока. Эти характеристики позволяют моделировать произвольно большие явления, где требуется компромисс между размерностью моделирования и уровнем детализации для поддержания производительности.

Существует несколько методов, основанных на подходе Лагранжа.

Самый простой из них – метод невзаимодействующих частиц (рисунок 3, слева), т. е. при моделировании не учитывается взаимодействие частиц и влияние на них внешних сил. Широко используется в компьютерной графике, поскольку требует значительно меньшего времени вычислений чем в других случаях. Этот тип симуляции может использоваться для создания вспомогательных эффектов воды, таких как водяные брызги и водяная пена.

Метод гидродинамики сглаженных частиц (SPH) был изобретен в области вычислительной астрофизики, но широко использовался в области гидродинамического моделирования. SPH — это в основном метод интерполяции, используемый для аппроксимации свойств жидкости по частицам в пространстве (рисунок 3, справа). Другими словами, SPH используется для аппроксимации непрерывных свойств жидкости путем интерполяции значений между дискретными положениями. Процесс интерполяции значений между частицами называется сглаживанием и осуществляется с помощью так называемых ядер сглаживания.

Во многих лагранжевых симуляциях используются две версии систем частиц: одна для имитации брызг и пены, и одна для имитации жидкости.

Частицы генерируются до начала программы и/или одним/несколькими эмиттерами. Эти излучатели могут иметь любую форму, но самыми простыми являются точечные и прямоугольные излучатели. Эмиттер обычно генерирует

частицы с рядом параметров, таких как скорость, масса, время жизни и т. д. Сам эмиттер также имеет ряд параметров, включая скорость генерации, силу импульса появления и т. д.

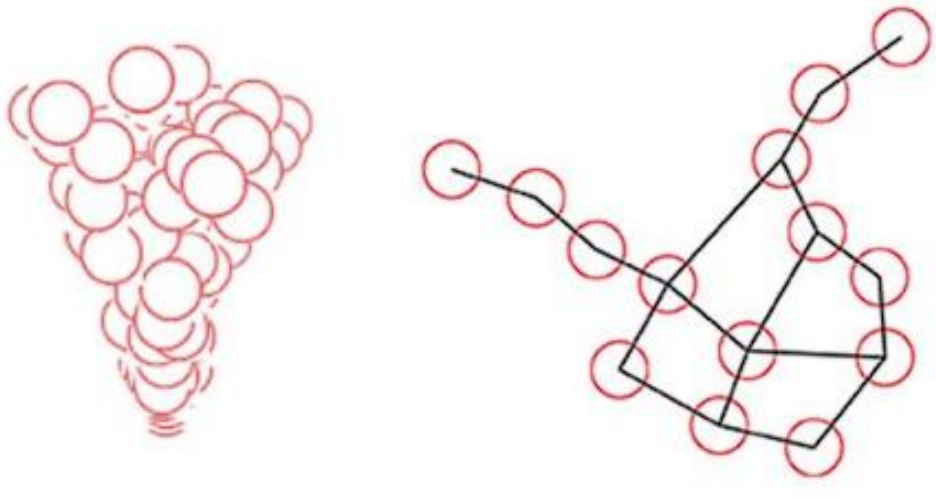


Рисунок 3 – Метод невзаимодействующих частиц (слева) и метод сглаженных частиц (справа)

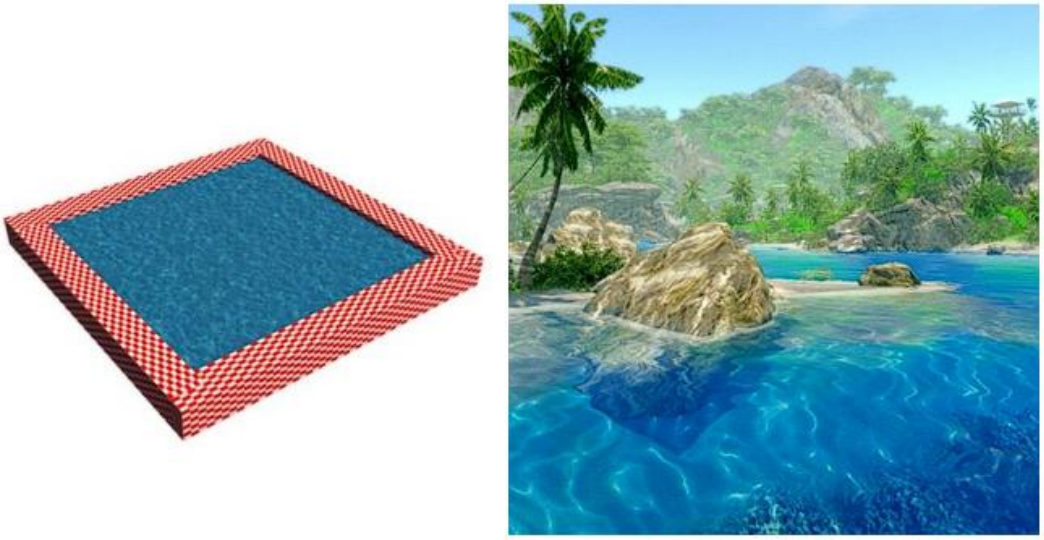


Рисунок 4 – Текстура воды, наложенная на плоскость

Еще один подход к моделированию жидкости – процедурная вода.

Процедурная вода — это термин, часто используемый для моделирования жидкости, в котором важен только визуальный эффект. Этот тип симуляции является самым быстрым, поскольку он заключается лишь в наложении текстуры на поверхность (рисунок 4). Обычно эффект движения жидкости достигается

путем смещения и масштабирования текстуры, изменения положения вершин меша поверхности. Метод часто используется для моделирования крупных бассейнов с жидкостью, таких как озера и океаны. Частным случаем этого подхода является моделирование на основе поля высот.

Для моделирования был выбран метод SPH, по следующим причинам:

– более простая реализация алгоритма регистрации и обработки столкновений с твердыми объектами, так как обрабатывать их можно на уровне отдельной частицы;

– проще определять свободные поверхности жидкости (за границу потока можно принять область потока с низкой плотностью);

– гибкость метода. Данный метод может также использоваться для симуляции деформируемых, сыпучих, газообразных тел;

– проще алгоритм расчета параметров. Так как значения рассчитываются для каждой частицы, этот процесс может быть распараллелен;

– вычислительные оптимизации. Например, при условии постоянного числа частиц не требуется дополнительные расчеты для выполнения закона сохранения энергии. Не требуется рассчитывать параметры массы жидкости, так как значение массы постоянно для каждой частицы;

– отсутствие необходимости разработки сложных алгоритмов для построения адаптивной сетки для динамичных процессов и объектов со сложной геометрией;

– более простое описание границ разделов сред и свободных границ, чем в сеточных;

– Возможность простого описания участков с высокой плотностью взаимодействующих частиц. Как указано в [11], в сеточных методах для решения подобных задач необходимо использовать динамически перестраиваемую неравномерную сетку, сильно сгущающуюся в области локализации объектов;

– простое описание взаимодействующих многокомпонентных сред с источниками и стоками, в том числе систем с фазовыми переходами вещества;

– возможность эффективного распараллеливания алгоритма расчетов.

1.4 Обзор существующих решений

Моделирование жидкости является относительно новым направлением в компьютерной графике. Его история развития насчитывает немногим более 30 лет.

Первая попытка моделирования жидкости была предпринята компанией NextLimitTechnologies в 1998 году, для чего была разработана технология Real-Flow. Несмотря на низкую вычислительную мощность компьютеров того времени удалось получить довольно реалистичную визуализацию. Преимущество данной технологии заключалась в том, что ранее использовался метод деформации твердого тела. В RealFlow же использовался метод SPH для представления поверхности жидкости в виде большого числа частиц, что позволило учесть мелкие детали (всплески, брызги, пену волн, пузырьки и т. д.). В дальнейшем технология активно развивалась, RealFlow стал популярным инструментом для создания компьютерной графики и спецэффектов.

На данный момент RealFlow поддерживает интеграцию со всеми основными 3D-редакторами: Maya, 3ds Max, Houdini, LightWave, Softimage. С 2015 появилась возможность использовать движок RealFlow прямо внутри редактора Cinema 4D с помощью дополнения RFCORE.

В 2015 году компанией Autodesk была разработана собственная система симуляции жидкостей – Bifrost. Это система создания высококачественных эффектов для жидкостей с использованием решателя FLIP (жидкие неявные частицы). Имеется возможность генерировать жидкость из эмиттеров, настраивать ее взаимодействие с гравитацией, расставлять коллайдеры чтобы направлять потоки и вызывать брызги, создавать узкие струи жидкости и другие эффекты. Для настройки моделирования система имеет удобный визуальный редактор.

В редакторе Blender для симуляции жидкости используется свой собственный воксельный движок FLIP Fluids, основанный на решении решеточных уравнения Больцмана. Его преимуществом является возможность моделировать движение вязких жидкостей, хотя движок ориентирован на воду.

С 2020 года в Blender стало доступно использование нового движка

Mantaflow. В отличие от FLIP Fluids, он позволяет моделировать не только течение жидкостей, но и движение газов. Движок также основан на решении решетчатых уравнений Больцмана.

Однако нужно учитывать, что описанные решения не ориентированы на высокую точность моделирования и не учитывают всех характеристик жидкостей, так как предназначены в первую очередь для создания визуальных эффектов. Поэтому в научных целях обычно используется специальное программное обеспечение.

Среди наиболее популярных программ можно выделить Ansys Fluent – программное обеспечение, которое позволяет эффективно выполнять расчеты для различных физических моделей жидкостей, например стационарного или переходного течения, несжимаемых или сжимаемых, ламинарных или турбулентных потоков, ньютоновских или неньютоновских жидкостей, идеального или реального газа.

Оно основано на методе конечных объемов, в котором область течения разделяется на конечное множество контрольных объемов; в этом множестве контрольных объемов решаются уравнения сохранения массы, импульса, энергии и т. д.; уравнения в частных производных дискретизируется в систему алгебраических уравнений; затем производиться численное решение этих алгебраических уравнений в расчетной области.

Также следует упомянуть Autodesk CFD – это CAE-система, предназначенная для расчетов и моделирования движения потоков жидкостей и газов, а также процессов теплопередачи и тепломассообмена. В ней реализована поддержка моделирования свободного течения жидкостей методом свободной поверхности VOF (Volume of Fluid).

Среди открытых проектов можно выделить следующие существующие решения для моделирования жидкости, которые могли бы использоваться в области разработки игр:

SPlisHSPlasH — это библиотека с открытым исходным кодом для физического моделирования жидкостей. Моделирование в этой библиотеке основано на

методе гидродинамики сглаженных частиц (SPH). В библиотеке используются современные решатели давления (WCSPH, PCISPH, PBF, IISPH, DFSPH, PF) для моделирования несжимаемости. Кроме того, библиотека предоставляет различные методы моделирования вязкости, поверхностного натяжения и завихренности. Результат работы библиотеки можно увидеть на рисунке 5.

Библиотека не интегрирована в среду разработки Unity. Может работать как на центральном, так и на графическом процессоре. Распространяется в виде исходного кода и для установки требует компиляции проекта с помощью CMake. Написана на C++, но существуют привязки для языка Python. Для работы необходимо создать файл формата json, с заданными параметрами моделирования. Имеет несколько встроенных инструментов: для семплирования 3D моделей (разбиения 3D модели на частицы), экспорта данных моделирования, генерации брызг на этапе постобработки, деформации мешей на основе данных моделирования.

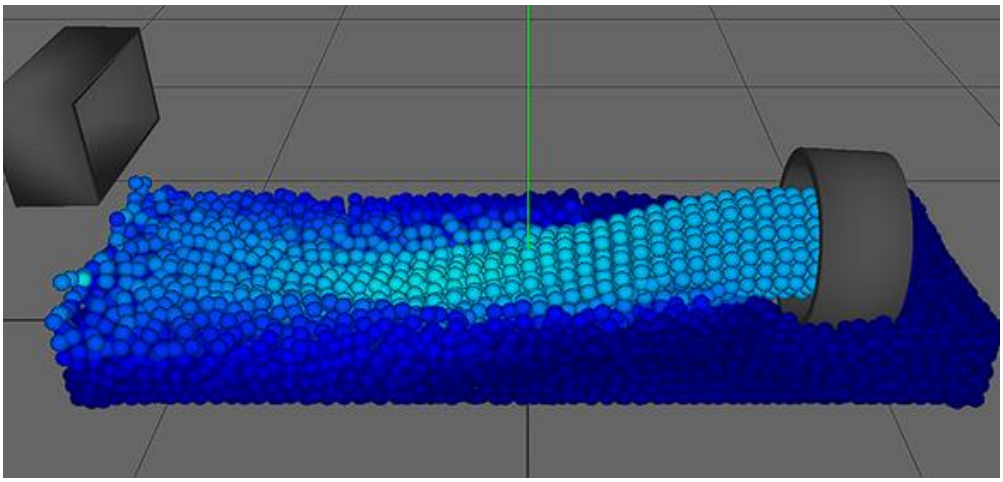


Рисунок 5 – Результат работы библиотеки

Position-Based-Dynamics – Проект по симуляции физических явлений на библиотеке PositionBasedDynamics. Содержит примеры реализации на Unity симуляции движения твердых тел, деформируемых тел, ткани, жидкости.

Распространяется в виде проекта Unity. Целиком написан на C#. Для работы симуляции на сцене необходимо создать объект, содержащий параметры моделирования. Для расчетов используется метод PBD, когда на основе применения внешних сил рассчитываются новые позиции точек, а затем применяются

граничные условия, которые гарантируют, что смещение точек физически корректно. Вычисления производятся на одном потоке CPU что вызывает проблемы с производительностью при достаточно большом количестве частиц. Не содержит дополнительных инструментов для импорта данных и семплирования моделей. Не содержит графических шейдеров для реалистичного отображения жидкости, вместо этого к каждой частице применен стандартный материал Unity. Результат работы представлен на рисунке 6.

Unity-Water-Simulation – Проект для симуляции воды. Распространяется в виде проекта Unity. Код написан на C#. Вода представляет собой меш, который обновляется каждый кадр за счет того, что каждой вершине присваивается новое значение, вычисленное на основе математического шума и смещения, увеличивающегося каждый кадр, что позволяет имитировать движение поверхности. Можно задать параметры, влияющие на перерасчет вершин и, следовательно, на отображение.

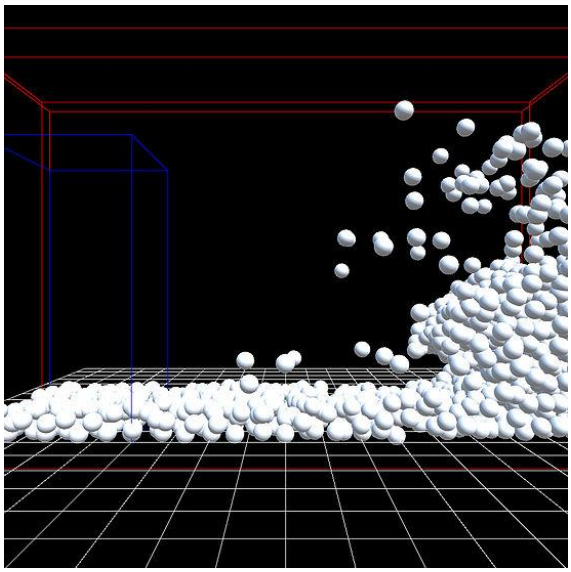


Рисунок 6 – Симуляция воды в проекте Position-Based-Dynamics

Для визуализации поверхности создан стандартный материал Unity, с заданными параметрами цвета, прозрачности и т. д (с поддержкой вычислений на основе освещения в сцене). Также написан ряд скриптов для обработки поведения объектов на воде (например учет силы Архимеда). Оптимизация выполняется встроенными средствами Unity, проблем с производительностью не наблюдаются. Однако такой подход подойдет только для моделирования статичного

объема жидкости. Результат работы представлен на рисунке 7.

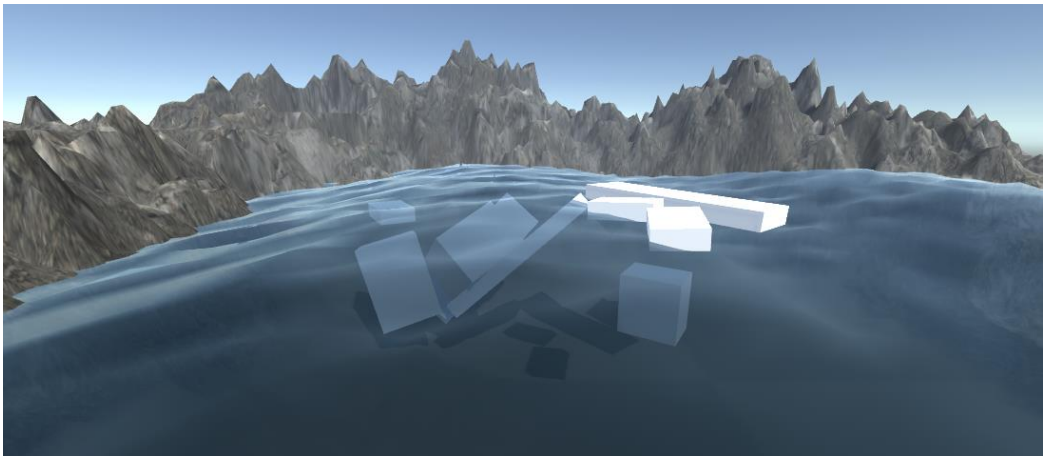


Рисунок 7 – Симуляция воды в проекте

Таблица 1 – Сравнение существующих библиотек

Название	Поставка и установка	Метод моделирования	Настройки	Недостатки
Unity Water Simulation	Распространяется в виде проекта Unity	Преобразование меша на основе шума	Настройка параметров шума и материала (влияет только на отображение)	Моделирование возможно только для статичного объема
Position Base Dynamics	Распространяется в виде проекта Unity	PBD (Расчет позиций каждой точки потока и применение условий ограничения)	Можно задать число частиц, идеальную плотность жидкости	Низкая производительность из-за выполнения на одном потоке CPU
SPlisHSPlasH	Распространяется в виде исходного кода и требует компиляции	SPH	Различные аспекты моделирования (расчет вязкости, завихренности, сопротивления и т.д.)	Отсутствует интеграция с Unity

Исходя из анализа аналогов можно сформулировать следующий предварительный набор требований к разрабатываемому программному продукту:

– должна поддерживаться интеграция со средой разработки Unity;

– должна быть возможность интерактивно настраивать параметры моделирования;

– должна поддерживаться достаточно реалистичная визуализация жидкости;

– достаточно высокая производительность (не менее 30 кадров в секунду, чего можно добиться путем использования параллельных вычислений);

– моделируемый поток жидкости должен взаимодействовать с объектами в сцене;

– отсутствие возрастающих ошибок с увеличением времени симуляции.

2 АЛГОРИТМИЧЕСКОЕ РЕШЕНИЕ ЗАДАЧИ

2.1 Описание математической модели

Большинство методов вычислительной гидродинамики основаны на уравнениях Навье-Стокса (формула 1) для несжимаемого и однородного потока. То что жидкость однородна означает что ее плотность не меняется по всему объему жидкого тела, а то что она несжимаема означает что плотность частицы не меняется во времени.

Основные этапы общего метода гидродинамики сглаженных [12]:

– системой уравнения в частных производных задаются законы движения среды;

– непрерывно аппроксимируются поля величин, описывающие среду, и пространственные дифференциальные операторы;

– полученная аппроксимация дискретизируется по отдельным частям;

– на основе полученной аппроксимации исходная система уравнений в частных производных преобразуется в систему обыкновенных дифференциальных уравнений, которая описывает динамику изменения рассматриваемых величин в отдельных частицах;

– полученная система решается численно с помощью какого-либо метода;

– полученные значения полей в частицах могут быть интерполированы.

Первое уравнение, уравнение импульса описывает как силы, действующие на жидкость, заставляют ее ускоряться:

$$\begin{gathered}\frac{\partial \vec{v}}{\partial t} = -(\vec{v}\cdot\nabla)\vec{v} - \frac{1}{\rho}\nabla P + \frac{\mu}{\rho}\nabla\cdot\nabla\,\vec{v} + F, \\ (\vec{v}\ \cdot\nabla)\vec{v} \equiv \left[v_x\frac{\partial v_x}{\partial x}, v_y\frac{\partial v_y}{\partial y}, v_z\frac{\partial v_z}{\partial z}\right] \\ \nabla P \equiv \left[\frac{\partial P}{\partial x}, \frac{\partial P}{\partial y}, \frac{\partial P}{\partial z}\right] \\ \nabla^2\vec{v} \equiv \left[\nabla^2 v_x, \nabla^2 v_y, \nabla^2 v_z\right], \nabla^2 v_x \equiv \frac{\partial^2 v_x}{\partial x^2} + \frac{\partial^2 v_x}{\partial y^2} + \frac{\partial^2 v_x}{\partial z^2},\end{gathered} \tag{1}$$

где $\vec{v}$ – это векторное поле скоростей. Скорость частицы в точке $x =$

$\left[x_x\ x_y\ x_z\right]^T$ в момент времени t равна $v = v(t,x) = [u(t,x)\ v(t,x)\ w(t,x)]^T$.

ρ – плотность жидкости в точке (постоянна).

P – Давление. Это скалярное поле, показывающее с какой силой жидкость действует на единицу площади

μ – коэффициент динамической вязкости. Показывает, насколько сильно жидкость сопротивляется деформации.

F – Внешние силы, действующие на единицу объема. Их называют объемными силами, так как эти силы воздействуют на все тело жидкости, а не только на поверхность. Часто приравнивается к ρg, где g — ускорение свободного падения.

Также необходимо учитывать уравнение неразрывности (формула 2):

$$\frac{\partial \rho}{\partial t} + \nabla(\rho \vec{v}) = 0, \tag{2}$$

где ρ – плотность жидкости в точке, $\vec{v}$ – векторное поле скоростей, t – момент времени.

Это уравнение представляет собой частный случай закона сохранения энергии и позволяет сделать следующее упрощение $\nabla \vec{v} = 0$. Учитывая это можно из формулы 1 получить уравнения движения конкретной частицы:

$$\frac{d\overrightarrow{v_i}}{dt} = \vec{g} - \frac{1}{\rho_i}\nabla P_i + \frac{\mu}{\rho_i}\nabla^2 \overrightarrow{v_i}, \tag{3}$$

где v – скорость частицы, t – момент времени, g – ускорение свободного падения, ρ – плотность частицы, μ – коэффициент вязкости, P – давление частицы.

Следующим шагом метода SPH является аппроксимация частиц. Как указано в [13], для этого жидкость представляется в виде множества частиц, которые полностью определяют состояние системы. При вычислении значений поля, частицы используются для выполнения операции интегрирования, дифференцирования, интерполирования.

При аппроксимации полагается, что каждая конкретная частица описывает определенный объем пространства V_i (подобно тому, как в механике в

материальной точке концентрируется масса). Учитывая массу и объем самой частицы, объем V_i , представляющий сконцентрированную в частице жидкость, можно представить как $\frac{m_i}{p_i}$.

То есть метод SPH позволяет провести интерполяцию значений и тем самым вычислить составляющие уравнения (3) численными методами. Он предполагает, что величина A_i в произвольном положении r приблизительно вычисляется с помощью набора известных величин A_j соседних частиц r_j:

$$A_i(r) = \int A(r')W(r - r', h)dr' \approx \sum_j \frac{m_j}{\rho_j} A(r_j)\, W(r - r_j, h), \quad (4)$$

где W– весовая функция, называемая ядром сглаживания, h – длина сглаживания, r – координаты текущей частицы, r_j – координаты соседней частицы.

В целях оптимизации учитывается только взаимодействие частиц в пределах расстояния, называемого длинной сглаживания. Стоит учесть, что длина сглаживания может выбираться в зависимости от плотности частиц (рисунок 8) так, чтобы содержать примерно одинаковое их количество. То есть должно выполняться условие:

$$W(r - r', h) = 0, \text{при } |r - r'| > kh, \quad (5)$$

где k зависит от функции W, а kh определяет ненулевую область сглаживающей функции.

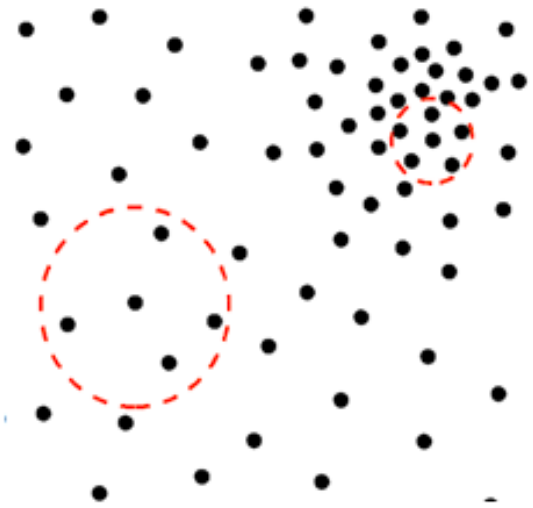


Рисунок 8 – Длина сглаживания в зависимости от плотности

Ядро сглаживания определяет, насколько сильно соседняя частица влияет на текущую, в зависимости от расстояния между ними. Общий вид такой функции представлен на рисунке 9.

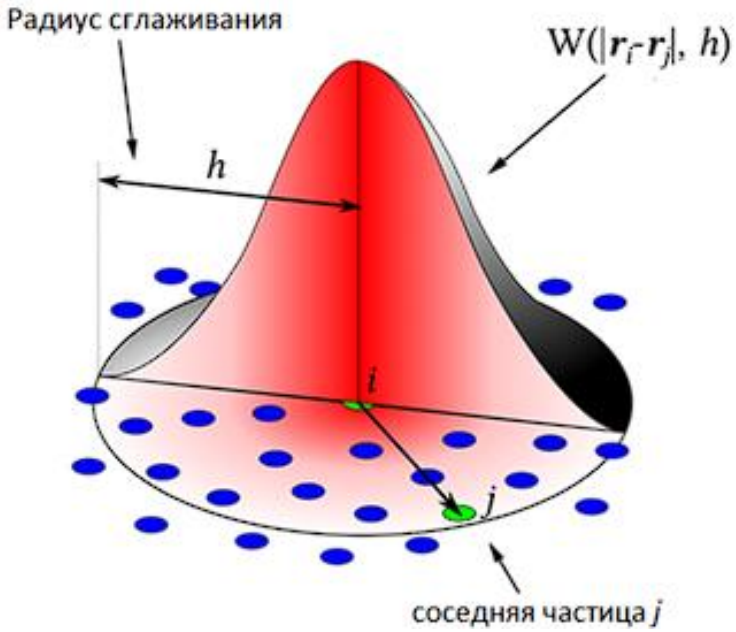


Рисунок 9 – Весовая функция

Ядро сглаживания выбирается произвольно, но оно должно удовлетворять следующим условиям:

$$1 = \sum_{j=1}^{N} W_{ij}$$
$$A_i = \sum_{j=1}^{N} A_i W_{ij} \quad (6)$$

Стабильность, точность и быстродействие метода SPH сильно зависят от выбора сглаживающих ядер. Кроме того, ядра, равные нулю с нулевыми производными на границе, способствуют устойчивости. Помимо этих ограничений, можно создавать ядра для специальных целей. В данной работе выбрано следующее ядро сглаживания:

$$W(r - r_j, h) = \frac{315}{64\pi h^9} \begin{cases} \left(h^2 - (r - r_j)^2\right)^{3,0} \leq r \leq h \\ 0 \end{cases}$$
$$\nabla W(r - r_j, h) = \frac{-45}{\pi h^6} \frac{r - r_j}{\|r - r_j\|} \begin{cases} \left(h - \|r - r_j\|\right)^2, 0 \leq r \leq h \\ 0 \end{cases} \quad (7)$$

$$\nabla^2 W(r - r_j, h) = \frac{45}{\pi h^6} \begin{cases} (h - \|r - r_j\|), 0 \le r \le h \\ 0 \end{cases}$$

Учитывая данную аппроксимацию, можно вычислить численными методами значения составляющих правой части уравнения Навье-Стокса.

Давление p можно вычислить с помощью уравнения Пуассона, однако это потребует больших вычислительных затрат. В метод SPH давление обычно вычисляется с помощью уравнения состояния идеального газа, то есть $p = kp_1$ либо одной из его модификаций, например:

$$p = k(\rho - \rho_0), \tag{8}$$

где k ⬿ коэффициент жесткости, ρ_0 – плотность жидкости в начальном состоянии, ρ – плотность частицы, для которой вычисляется давление. Это уравнение обычно используется для газов. Для несжимаемых жидкостей лучше подходит формула:

$$p = \frac{\rho_0 c^2}{\gamma}\left(\left(\frac{\rho}{\rho_0}\right)^{\gamma} - 1\right) \tag{9}$$

где c – скорость звука в среде, либо максимально возможная скорость движения частицы, γ – коэффициент, обычно равный 7 для несжимающихся жидкостей.

На основе аппроксимаций оператора ∇ основывается множество подходов к вычислению давления. При этом необходимо чтобы воздействие частиц друг на друга было симметричным, то есть выполнялся третий закон Ньютона.

Силу давления F^{pres} для частицы i с учетом симметрии воздействия можно вычислить по формуле:

$$F_i^{pres} = m_j \sum_j \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2}\right) \nabla W_{ij} \tag{10}$$

где m – масса частицы, p – давление частицы, W_{ij} – значения ядра сглаживания, ρ – плотность частицы.

При стандартной SPH аппроксимации силу вязкости несжимаемой жидкости можно выразить как:

$$\nabla^2 v_i = \sum_j \frac{m_j}{\rho_j} v_j \nabla^2 W_{ij}, \tag{11}$$

где v_i – скорость частицы, ρ_j – плотность частицы, m_j – масса частицы, W_{ij} – значение ядра сглаживания. Однако согласно [14], такой подход приводит к тому что вторая производная ядра сглаживания меняет знак внутри длины сглаживания. Поэтому могут возникать ошибки интерполяции при вычислении положений частиц. Поэтому используется предложенный в [15] подход к вычислению вязкости с помощью метода конечных разностей:

$$\nabla^2 v_i = 2(d+2) \sum_j \frac{m_j}{\rho_j} \left(\frac{(v_i - v_j)(r_i - r_j)}{\|r_i - r_j\|^2 + 0.01h^2} \right) \nabla W_{ij}, \tag{12}$$

где d – число измерений в пространстве, r – координаты частицы, h – длина сглаживания. Параметр $0.01h^2$ используется чтобы избежать деления на ноль.

Уравнение плотности также выводится с помощью аппроксимации из уравнения неразрывности 2.

Сила поверхностного натяжения рассчитывается как внешняя по формуле:

$$F_{tension} = k \sum_j m_j (r - r_j) W(r - r_j, h) \tag{13}$$

где k – коэффициент поверхностного натяжения. Данная формула основывается на том, что на частицы, находящиеся на поверхности жидкости, сила действует только с одной стороны. Во внутреннем пространстве жидкости частицы в среднем расположены на одинаковых расстояниях и соответственно имеют равное число соседей, поэтому воздействие силы поверхностного натяжения близко к нулю.

При этом можно не учитывать силу поверхностного натяжения при моделировании масштабных сцен, например при моделировании океана.

Таким образом составляющие правой части уравнения Навье-Стока можно выразить следующими уравнениями:

$$\rho_i(r) \approx \sum_j m_j W(r - r_j, h) \tag{14}$$

где ρ – плотность частицы, m – масса частицы, r – координаты частицы, h

– длина сглаживания, W – ядра сглаживания.

$$\frac{\nabla P_i}{\rho_i} \approx \sum_j m_j \left(\frac{P_i}{\rho_i^2} + \frac{P_j}{\rho_j^2}\right) \nabla W(r - r_j, h) \quad (15)$$

где P – давление частицы.

$$\frac{\mu}{\rho_i} \nabla^2 v_i \approx \frac{\mu}{\rho_i} \sum_j m_j \left(\frac{v_j - v_i}{\rho_j}\right) \nabla^2 W(r - r_j, h) \quad (16)$$

где μ – коэффициент вязкости.

Для интегрирования используется схема предиктор-корректор. Обозначим вектор переменных $(r_i; \rho_i; v_i)$ как X, а значение правых частей уравнений как F, то есть $F^n = F(X)$. Тогда на шаге предсказания получаем значение средней точки X: $X^{n+\frac{1}{2}} = X^n + \frac{F^n \Delta t}{2}$, где Δt – временной шаг.

Далее на шаге корректора получим значение X в конце шага интегрирования: $X^{n+1} = X^n + F^{n+\frac{1}{2}} \Delta t$.

Тогда алгоритм расчета параметров частиц в каждый момент времени будет выглядеть следующим образом:

– сначала необходимо найти расстояния между всеми частицами. На этом этапе действует алгоритм поиска соседей;

– далее нужно вычислить плотность частицы ρ_i используя формулу (13);

– рассчитать давление частицы по формуле:

$$P_i = B\left[\left(\frac{\rho}{\rho_0}\right)^7 - 1\right], \quad (17)$$

где ρ – желаемая плотность покоя жидкости, B – константа жесткости. На практике большая константа жесткости снижает сжимаемость жидкости, но требует меньших временных шагов интегрирования;

– вычислить градиент давления $\frac{\nabla P_i}{\rho_i}$ по формуле 15;

– вычислить компонент вязкости $\mu \nabla^2 v_i$ по формуле 16;

– вычислить влияние внешних сил, в том числе гравитации, действий пользователя и т.д;

– вычислить сумму сил, действующих на частицу;

– численно проинтегрировать уравнения скорости и позиции:

$$u_{\mathrm{i}} = u_{\mathrm{i}} + \Delta t \frac{F_{\mathrm{i}}}{\rho_{\mathrm{i}}}$$
$$r_{\mathrm{i}} = r_{\mathrm{i}} + \Delta t\, u_{\mathrm{i}}, \tag{18}$$

где u – скорость частицы, t – момент времени, F – сумма сил, действующих на частицу, ρ – плотность частицы, r – координаты частицы;

– применить граничные условия.

Полученная модель учитывает воздействие внешних сил, вызванных гравитацией, столкновениями, взаимодействием с пользователем. Эти силы приложены непосредственно к частицам.

На основе данной модели можно определить следующий порядок действий симуляции жидкости:

Сначала на сцене необходимо задать пространство моделирования жидкости и области представляющие граничные условия. Также нужно задать параметры моделирования, такие как длина сглаживания, плотность жидкости, вязкость, радиус частиц и т. д.

Далее обозначенные выше пространства заполняются частицами, в соответствие с указанными параметрами моделирования (в частности радиусом частицы, плотностью жидкости).

Далее начинается итерационный процесс расчета значений частиц. Сначала выполняется алгоритм поиска соседей, потом для каждой частицы рассчитываются новый значения плотности, давления, силы вязкости. Затем с помощью численного интегрирования выполняется расчет скоростей и координат частиц.

Чтобы обеспечить достаточную точность вычислений необходимо использовать значительное число частиц, что сопряжено с большим объемом вычислений. Чтобы повысить производительность приложения данные расчеты должны выполнять с использование параллельных алгоритмов. Однако стоит учесть, что при выполнении параллельных вычислений на CPU и GPGPU имеются свои особенности.

Для CPU обычно используется технология параллельных вычислений MPI

/ OpenMP, благодаря чему нет большой проблемы в передачи данных между потоками.

В свою очередь в GPGPU архитектурах память исполняющегося потока обычно мала. Поэтому приходится часто выполнять обмены между различными типами памяти что является трудозатратной операцией. Но это компенсируется тем, что количество потоков на GPU значительно больше, чем может выполнять CPU, что обеспечивает намного большую скорость вычислений.

2.2 Описание решения задачи поиска соседей

Суть алгоритма поиска соседей заключается в том, чтобы для каждой частицы определить набор «соседей», которые оказывают на нее влияние. При этом соседняя частица должна находиться от рассматриваемой на расстоянии не более величины kh, где k – коэффициент, зависящий от выбранного ядра сглаживания, а h – длина сглаживания. Выбор оптимального алгоритма поиска соседей очень важен, так как от него во многом зависит скорость вычислений.

Рассмотрим наиболее популярные подходы к решению задачи поиска соседей:

Самым простым в реализации является метод прямого перебора частиц. Он предполагает вычисление расстояния между всеми частицами и выбор тех, у которых оно оказалось меньше величины kh. Принцип работы данного метода изображен на рисунке 10.

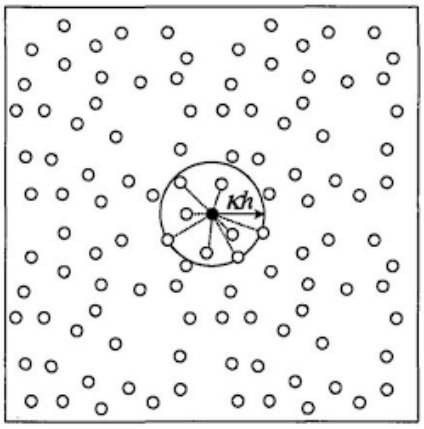


Рисунок 10 – Метод прямого перебора частиц

Хотя данный метод не подходит для задач с большим количеством частиц,

из-за своей неэффективности и ресурсоемкости, он может применяться для проверки работы более сложных алгоритмов.

Метод использования вспомогательной сетки основывается на принципе пространственного хещирования частиц. Данный метод предполагает наложение на пространство моделирования вспомогательной сетки со стороной ячейки равной величине kh (рисунок 11). Стоит отметить, что используемая сетка зависит лишь от размеров пространства моделирования и не зависит от его частной геометрии. Также ячейки вспомогательной сетки не используются для аппроксимации значений.

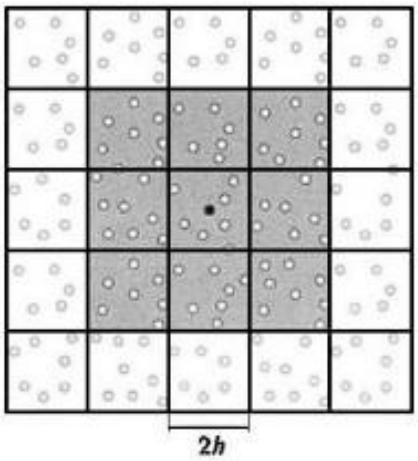


Рисунок 11 – Метод вспомогательной сетки

Хеширование в данном методе означает, что на каждой итерации определяется к какой ячейке принадлежит частица. Частицы, расположенные в одной и той же или соседних ячейках можно считать соседями.

Метод построения октодерева предполагает рекурсивное разбиение пространства моделирования на октанты (рисунок 12). Для этого в центре исходного пространства выбирается опорная точка, вокруг которой строятся дочерние октанты. Далее разбиение повторяется для каждого построенного октанта [16]. Разбиение прекращается если в заданном октанте не оказалось частиц, или размер ячейки оказался меньше величины kh.

Одним из недостатков данного метода является плохая адаптивность. Например, при нахождении жидкости в спокойном состоянии в ячейках

образуется много пустого пространства (в то время как оно должно по возможности сразу же отбрасываться), что замедляет скорость вычислений.

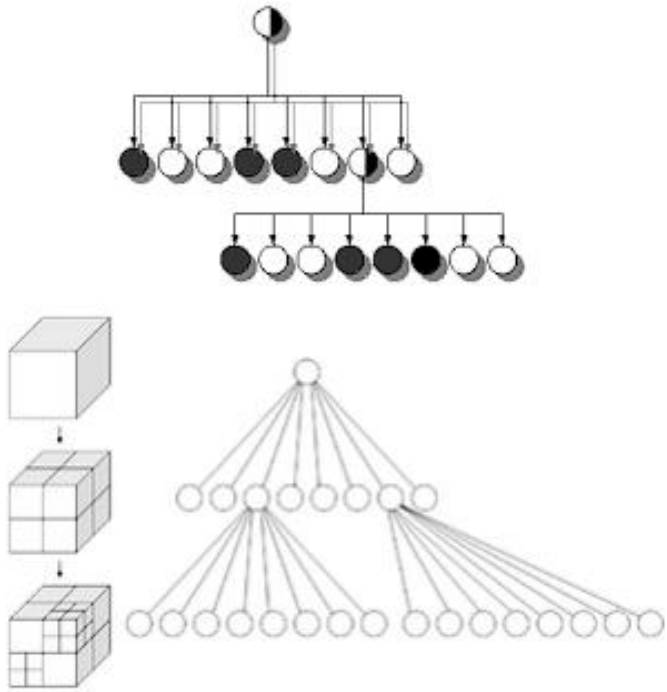


Рисунок 12 – Построение октодерева

В информатике и вычислительной геометрии kd-деревья стали популярной структурой данных, используемой для организации точек в K-мерном пространстве, где K обычно очень большое число. Это связано с тем, что эти структуры обеспечивают эффективный поиск точек в многомерном пространстве.

kd-дерево — структура бинарного пространственного разбиения. Эта структура представляет собой бинарное дерево ограничивающих параллелепипедов, вложенных друг в друга.

Kd-дерево строится рекурсивно по следующему принципу (рисунок 13): пространство моделирования разбивается медианной плоскостью на две ячейки, содержащие примерно одинаковое число частиц. Далее каждая из этих ячеек также разбивается медианной плоскостью, но перпендикулярной другой оси координат. Данный процесс повторяется пока в ячейке не останется одна частица.

В данной работе для поиска частиц использовался метод хеширования частиц с использованием вспомогательной сетки. Данный алгоритм был выбран из-

за простоты реализации при достаточно высоком быстродействии. Для вычисления функции хэширования все пространство моделирования разбивается однородной сеткой на ячейки со стороной $s = (s^x, s^y, s^z)$.

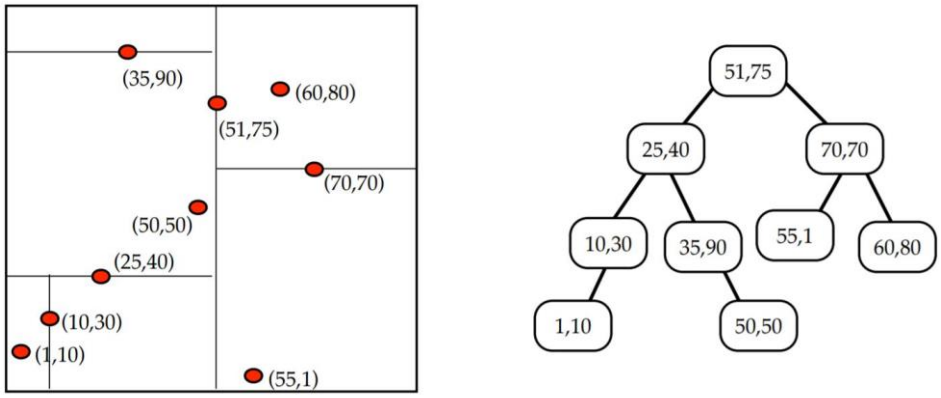


Рисунок 13 – KD деревья

Далее для каждой частицы необходимо рассчитать значения хеша и отсортировать по нему массив частиц. При этом несколько частиц могут попасть в одну ячейку, так как хэш функция может давать одинаковые значения если они расположены достаточно близко [17]. Таким образом частицы находящиеся рядом будут располагаться в массиве подряд. Сортировка массива частиц на каждом шаге моделирования требует временных затрат, однако по сравнению с прямым перебором удается снизить сложность алгоритма с O(N2) до O(cnN), cn, – среднее количество соседних частиц.

Поиск соседей выполняется параллельно несколькими потоками на GPU. Изначально массив частиц расположен в глобальной памяти GPU к которой каждый поток обращается для определения соседних частиц. Далее, чтобы уменьшить число обращений к глобальной памяти, найденные соседние частицы копируются в разделяемую память, совместно используемую блоком потоков, что позволяет избежать потерь производительности из-за переключения контекста.

2.3 Визуализация частиц

Этап визуализации течения жидкости при использовании GPU реализован с использованием технологии интероперабельности компьютерной графики. Координаты частиц постоянно находятся в памяти графического адаптера, поэтому

отображение этих данных выполняется без участия центрального процессора. Память, в которой расположены координаты частиц, отображается (передается как указатель) на память (буфер), из которого производится рисование.

Отрисовка объектов в Unity осуществляется с помощью набора этапов, известных как конвейер компьютерной графики. Для описания визуальных представлений используются базовые геометрические формы – вершины, ребра и грани. Вершина — это координата текстуры, которая может включать также информацию о цвете, векторе нормали и т. д. Ребро — это соединение между двумя вершинами. Грань — это замкнутое множество рёбер.

Во время отрисовки модели, ей данные обрабатываются так называемым графическим движком, который, в свою очередь, основан на низкоуровневом языке программирования. Основная идея графических процессоров состоит в том, чтобы обрабатывать несколько задач одновременно (параллельная обработка), уделяя основное внимание аппаратному ускорению вывода рендеринга. Для достижения этого результата данные геометрии модели передаются между следующими условными этапами (рисунок 14):

Сначала список всех вершин, определяющих геометрию модели, передается от CPU к GPU для обработки. Далее на GPU выполняется программа обработки вершин – набор инструкций API, позволяющих выполнять операции над вершинами (например трансформация позиций вершин, расчет освещенности, изменение текстурных координат). После выполняется этап сборки примитивов – на основе данных вершин и переданного списка индексов выполнятся создание каркасного представления модели. Также здесь отсекаются вершины, которые не должны отрисовываться в кадре, например задние полигоны.

На этапе растеризации происходит сопоставление собранных примитивов с пикселями экрана, называемых пикселы. В программе обработки фрагментов для полученного изображения вычисляются атрибуты пикселов. Например, если вершины треугольника окрашены разным цветом, цвет пикселов этого треугольника вычисляется интерполяцией цветов вершин треугольника, в зависимости от расстояния до каждой из 3-х вершин.

Последняя стадия конвейера применяет серию тестов на пикселы. Также на этом этапе обрабатывается прозрачность. Затем полученные данные передаются в буфер кадра. Пока одно визуализируемое изображение находится на экране, другое, называемое буфером кадра, заполняется подготовленными фрагментами, и когда этот процесс завершается, кадр обновляется.

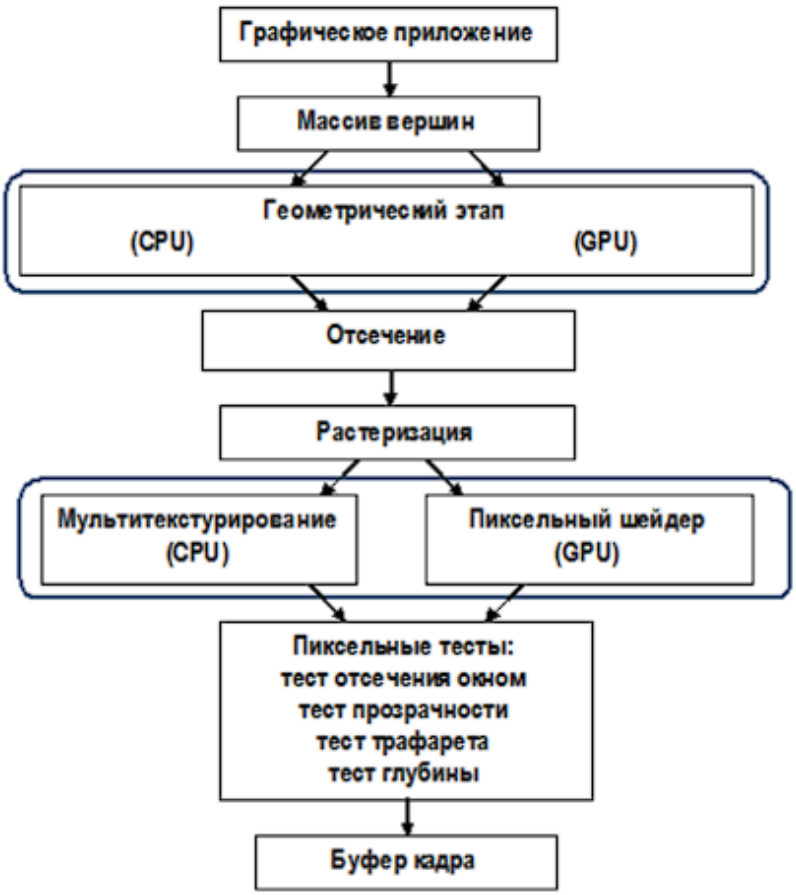


Рисунок 14 – Основные этапы работы графического конвейера

Современные видеокарты дают программисту возможность изменять воздействие двух из описанных выше стадий: вершинные шейдеры могут быть написаны для стадии вершинного преобразования и пиксельные шейдеры изменяют функционал стадии пиксельного текстурирования и окрашивания.

В вершинном шейдере можно запрограммировать такие задачи, как:

– изменение позиции вершины с использованием матриц проекции и моделей;

– трансформацию нормалей и, если нужно, их нормализацию (приведение к единичной длине);

– трансформацию и генерацию текстурных координат;

– освещение на вершину или рассчет данных для освещения на пиксел;

– операции с цветом.

Пикселный шейдер может выполнять следующие операции:

– вычисление текстурных координат и цветов на пиксел;

– применение текстуры;

– расчет тумана;

– вычисление нормалей, если нужно пикселное освещение.

Визуально частицу планируется представить как круговой градиент с размытыми краями, чтобы граница жидкости казалось прозрачной, а область скопления частиц выглядела более темной. Из-за ограниченной производительности мобильных устройств решено не учитывать отражения света от поверхности частицы. Градиент будет создан в виде процедурной текстуры. Смысл заключается в том, что вместо того, чтобы вручную создавать новые текстуры в графическом редакторе, можно программно сгенерировать двумерный массив пикселей и добавить их в новую текстуру. После этого текстуру нужно передать в шейдер, чтобы он мог использовать её для вычислений.

Эта техника обычно используется если нужно нарисовать динамически созданную текстуру на существующей текстуре при взаимодействии игрока с окружающим игровым миром. Например, с её помощью можно реализовать эффект разрушений на стене, следы на снегу или создавать процедурные фигуры, которые потом можно использовать в шейдере. Её использование необходимо, так как частицы не являются объектами сцены, а отрисовываются вручную, поэтому Unity не может применить к ним заранее созданный материал.

Шейдеры в Unity делятся на три большие категории:

– поверхностные шейдеры являются частью графического конвейера. Это наиболее распространенный тип. Они выполняют вычисления, определяющие цвет пикселей на экране. Они представлены файлами типа Shader;

– вычислительные шейдеры выполняют вычисления на графическом процессоре вне обычного графического конвейера. Они используются для расчета

показателей частиц на каждом шаге моделирования;

– шейдеры трассировки лучей выполняют вычисления, связанные с трассировкой лучей. В данном проекте не используются, так как обработка отражений для большого числа частиц вызовет недопустимое снижение числа кадров на мобильных устройствах.

Поверхностный шейдер – это оболочка, которая позволяет определять несколько шейдерных программ в одном файле и указывать Unity, как их использовать, а также содержит вспомогательную информацию, такую как имя шейдера, типы параметров, резервный шейдер, который будет использоваться если этот не доступен (например, из-за аппаратных ограничений). Объект shader должен иметь один или несколько блоков subshader (для реализации шейдера на разных платформах и оборудовании). Они содержит основной код шейдера и информацию о том, с каким оборудованием, конвейерами рендеринга и параметрами среды выполнения совместим этот SubShader. Внутри блока SubShader содержатся вспомогательные значения, такие как теги (пары ключ-значение).

Код шейдера записывается путем встраивания «фрагментов Cg/HLSL» в текст шейдера. Фрагменты затем компилируются в низкоуровневую сборку шейдера редактором Unity, а окончательный шейдер, включенный в файлы игры, содержит только эту низкоуровневую сборку или байт-код, зависящий от платформы.

Стоит уделить внимание процессу оптимизации. Каждый кадр процессор передает видеокарте инструкцию для отрисовки объекта. Причем такой вызов увеличивает нагрузку на CPU и GPU, так как инициирует ресурсоемкие операции по проверке и преобразованию данных графическим драйвером. По большей части это связано с изменениями состояния материала между вызовами отрисовки. Чтобы уменьшить их число в Unity существует встроенный инструмент, называемый material instancing. Он позволяет объединить несколько объектов с одинаковым материалом вместе, что позволит передать их вершины, параметры шейдера и т. д. на отрисовку за один вызов.

В Unity поверхностный и стандартный шейдеры поддерживают

группировку по материалу по умолчанию, достаточно лишь указать её использование в настройках материала. Однако существуют ограничения, связанные с количеством вершин и вершинных атрибутов (нормали, uv координаты и т. д.), которые не позволят передать на отрисовку все объекты сцены за раз. Тем не менее, использование группировки позволит достаточно сильно уменьшить число вызовов отрисовки.

Также нужно учитывать некоторые особенности при написании кода. Например, для передачи параметров из скрипта в шейдер часто используют обращению к свойству Material класса Renderer, отвечающему за отрисовку объектов. Но это свойство возвращает ссылку на материал, созданный для этого конкретного объекта. Таким образом для каждого объекта создается отдельный материал, что не позволит использовать группировку по материалу для отрисовки. Но если обращаться к свойству SharedMaterial, которое возвращает прямую ссылку на сам материал, параметры будут общими для всех группируемых объектов. Поэтому для передачи параметров в проекте используется специальный класс MaterialPropertyBlock. Также в коде шейдеров указывается, какие параметры могут быть объединены в буфер.

2.4 Обоснование выбора программно-технического обеспечения

В качестве среды разработки был выбран игровой движок Unity по следующим причинам:

– компонентно-ориентированный подход — разработчик прописывает объекту компоненты вроде возможности управления объектом и модели поведения;

– кроссплатформенность;

– условно-бесплатная модель распространения;

– большая библиотека ассетов и плагинов, которые можно использовать для создания прототипа и готовой игры;

– стабильность работы;

– широкие возможности по оптимизации;

– подробная документация, множество обучающих материалов, активное

сообщество.

В среде Unity для написания управляющих скриптов используется C#, поэтому именно он был выбран в качестве языка программирования.

C# – объектно-ориентированный язык программирования. Язык имеет статическую типизацию, поддерживает полиморфизм, перегрузку операторов (в том числе операторов явного и неявного приведения типа), делегаты, атрибуты, события, свойства, обобщённые типы и методы, итераторы, анонимные функции с поддержкой замыканий, LINQ, исключения, комментарии в формате XML.

Для написания кода был выбран текстовый редактор Visual Studio, так как он включает в себя все необходимые для разработки инструменты, такие как: отладчик, инструменты для работы с Git, подсветку синтаксиса, IntelliSense и средства для рефакторинга. Имеет широкие возможности для настройки: пользовательские темы, сочетания клавиш и файлы конфигурации.

В качестве физического движка выбран Havok Physics, так как предлагает самую быструю и надежную технологию обнаружения столкновений и физического моделирования, поэтому он популярен в игровой индустрии. Он основан на Unity Physics, физическом движке C#, написанном Unity и Havok для DOTS.

Этот пакет предоставляет серверную часть моделирования физики с закрытым исходным кодом, использующую тот же движок Havok Physics, который используется во многих ведущих в отрасли играх AAA. В этой реализации используются те же форматы входных и выходных данных, что и в Unity Physics, а это означает, что можно в любой момент просто поменять местами серверную часть симуляции, не изменяя какие-либо существующие физические активы или код.

По сравнению с симуляцией Unity Physics, Havok Physics for Unity предлагает:

– улучшенная производительность моделирования: Havok Physics для Unity – это движок с отслеживанием состояния, что означает, что время моделирования более чем в два раза быстрее, чем у Unity Physics, в сценах со значительным количеством твердых тел. Это связано с автоматическим засыпанием неактивных твердых тел и другими передовыми методами кэширования;

– более высокое качество симуляции: Havok Physics для Unity — это достаточно надежный движок, с множеством вариантов использования.

Глубокое профилирование и отладка физических симуляций с помощью визуального отладчика Havok (доступно только в Windows). Этот ведущий в отрасли инструмент может помочь определить подробные данные о многопоточной производительности в режиме реального времени, которые точно показывают, где происходят просадки производительности для всех ядер целевой системы.

2.5 Обзор возможностей профильного программного обеспечения

Графические функции Unity позволяют контролировать внешний вид приложения. Их легко настраивать для создания красивой, оптимизированной графики на различных платформах, от мобильных игр до консолей и настольных компьютеров.

К функциям по работе с графикой относятся:

– конвейер рендеринга – выполняет ряд операций (такие как отбраковка задних поверхностей, рендеринг, постобработка), которые извлекают содержимое сцены и отображают его на экране. Unity предоставляет три готовых конвейера рендеринга с разными возможностями и характеристиками производительности, но можно также создать свой собственный;

– система камер – позволяет определить видимые участки 3D сцены и обработать их для отображения (с учетом положения и ориентации камеры, перспективы);

– система освещения симулирует поведение света в реальном мире. Unity использует точные модели освещения для более реалистичного результата и упрощенные модели для более стилизованного результата. Поддерживаются прямое и непрямое освещение, освещение в реальном времени и статичное освещение, глобальное освещение, тени, отражения;

– наложение текстур. Текстуры применяются к объектам с помощью материалов. Материалы используют специализированные графические программы для рендеринга текстуры на поверхности сетки. Шейдеры могут реализовывать световые и цветовые эффекты для имитации блестящих или неровных

поверхностей среди многих других вещей. Они также могут использовать две или более текстур одновременно, комбинируя их для еще большей гибкости;

– предоставление готовых компонентов пользовательского графического интерфейса;

– система частиц для отображения таких эффектов как пламя, дым, искры и т. д.;

– поддержка карты высот местности;

– поддержка разных видов шейдеров (графических, вычислительных, трассировки лучей);

– применение эффектов постобработки;

– поддержка skybox (тип фона, который камера рисует до того, как отрисует кадр);

– работа с разными цветовыми пространствами;

– поддержка графических API DirectX, Metal, OpenGL и Vulkan в зависимости от доступности API на конкретной платформе;

– предоставление инструментов профилирования и оптимизации графики.

Физический движок Havok Physics предоставляет богатые возможности для симуляции. Он специально был разработан для удовлетворения требований к производительности динамичных игр, которые часто включают сложные сцены с большим количеством физических взаимодействий. Он обеспечивает стабильное наложение физических тел, минимальные артефакты при наличии быстро движущихся тел и в целом более контролируемое поведение, особенно при наличии неоптимизированной геометрии столкновений. Движок позволяет решать статические/динамические взаимопроникающие твердые тела за несколько кадров.

Движок предоставляет следующие возможности:

– настройка управления персонажем на основе физики для игры от первого и третьего лица;

– применение настраиваемых физических компонентов к игровым объектам;

– использование коллайдеров для обнаружения коллизий между объектами;

– поддержка настраиваемых шарнирных соединений объектов сцены;

– конфигурирование сложных системы твердых тел и соединений, организация их в виде логического дерева, в котором каждое отношение родитель-потомок отражает взаимно ограниченное относительное движение;

– симуляция движения ткани одежды и другого текстиля;

– управление различными физическими контекстами в одном проекте с несколькими выделенными физическими сценами.

Движок включает инструмент отладки – Havok Visual Debugger (VDB). Подключаясь через TCP/IP к приложению, он позволяет удаленно диагностировать сложное поведение, проблемы с производительностью или сбои. При подключении приложение предоставляет пользователю VDB множество настроек в зависимости от проблемы, которую он пытается решить. После настройки приложение создает сервер, предлагающий информацию подключенным клиентам VDB.

VDB позволяет:

– сделать паузу и перематывать время назад, чтобы увидеть предыдущие взаимодействия в симуляции;

– изучать статистику производительности как в графическом, так и в текстовом виде;

– видеть коллайдеры, из которых состоят твердые тела в сцене;

– видеть соединения и контакты между телами;

– видеть статус (де)активации динамических тел;

– посмотреть тепловую карту моделирования, чтобы точно определить важные области, требующие немедленной оптимизации;

– взаимодействовать с физической симуляцией напрямую.

Клиент VDB также может записывать и загружать предыдущие снимки моделирования для просмотра. Эти снимки могут быть полезны, когда разработчикам нужна внешняя поддержка для отладки сложных физических сценариев.

3 ПРОГРАММНОЕ РЕШЕНИЕ ЗАДАЧИ

3.1 Описание архитектуры проекта

Входными данными программы являются параметры моделирования, а именно материал, применяемый к частицам жидкости, положения граничных точек, размер частиц, эталонная плотность жидкости, материал сглаживания (влияет на визуализацию жидкости), константы, используемые для проведения расчетов и влияющие на аппроксимацию результатов (такие как коэффициент динамической вязкости, размер ячейки частицы, количество итераций на разных этапах вычислений). Также, чтобы обеспечить динамическое взаимодействие с потоком среда Unity должна передавать в систему моделирования данные о положении объектов в сцене.

Из-за того, что решение использует вычислительные шейдеры, к целевым платформам предъявляются следующие требования:

- PC(DX11/DX12/OpenGL/Vulkan, x86 and x64);
- UWP(DX11/DX12, x86 and x64);
- macOS(Metal, x64 and arm64);
- iOS (iOS version 12 or later, iPhone 6s or newer);
- Android (OpenGL ES 3.2/Vulkan, armv7 and arm64, Android 7.0 or later, Oculus Quest devices are not supported);
- PC VR.

Стоит также отметить, что приложение должно работать при достаточно высокой частоте кадров. Частота кадров, измеряемая в кадрах в секунду (FPS), – это термин, часто используемый для определения плавности работы игры на компьютере. Если время рендеринга изображения задано в миллисекундах, то частота кадров – это просто сумма изображений, отрисованных за полную секунду. Для современных компьютерных игр целевая частота кадров должна составлять не менее 30–60 FPS. Частота кадров 30–60 FPS означает от 33.33 до 16.67 миллисекунд для рендеринга каждого кадра.

Учитывая, что большая часть вычислительной мощности необходима для

игровых механик и процессов, это оставляет лишь небольшое количество вычислительной мощности для симуляции жидкостей. Следовательно, необходимо выполнить следующие требования к симуляции:

– она должна выглядеть реалистично;

– она должна быть интерактивной;

– она должна потреблять мало памяти;

– она должна быть стабильной.

– симуляция должна работать со скоростью не менее 30 кадров в секунду.

Программа состоит из следующих модулей (рисунок 15):

– модуль инициализации предоставляет пользователю графический интерфейс для ввода начальных условий (размер и количество частиц, их начальные координаты, граничные условия и т.д.), параметров расчета и передает эти данные в другие модули. Также запускает процесс моделирования, а по завершении освобождает ресурсы;

– модуль обработки столкновений определяет столкновения частиц с объектами сцены и передает эти данные в модуль расчетов;

– модуль расчетов, учитывая применение граничных условий и воздействие внешних сил, расчитывает новые координаты чатиц;

– модуль сортировки используется для оптимизации самой затратной чатси алгоритма расчетов – поиска соседей частицы. Он выполняет хеширование и сортировку частиц и передает эти данные в модуль расчетов;

– модуль визуализации отвечает за рендеринг поверхности жидкости. Для этого он на основе данных визуализации (освещенность сцены, плотность частиц в точке, угол наклона полигона в точке, длина пути светового луча, параметры материала частиц и т.д.) генерирует текстуру, которая применяется на специальном объекте рендеринга. Таким образом частицы которые находятся внутри это объекта могут быть визуализированы.

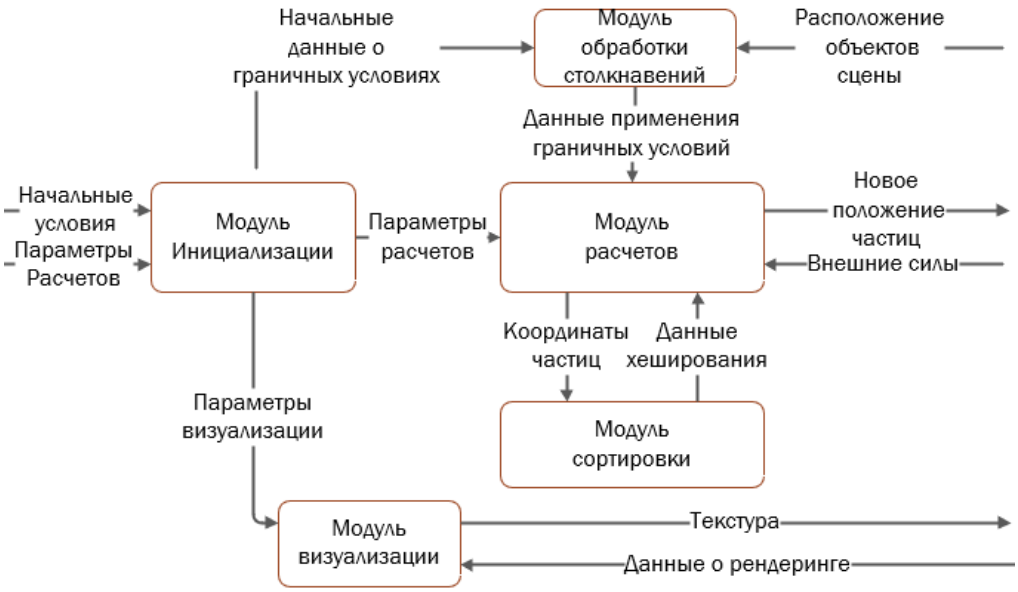


Рисунок 15 – Взаимодействие модулей программы

Для демонстрации работы алгоритма было создана тестовая сцена со следующими объектами:

– главная камера. Отвечает за отображение объектов сцены. Также имеет скрипт управления, для того чтобы можно было перемещать камеру во время проигрывания сцены;

– объект освещения. Чтобы рассчитать затенение 3D-объекта, Unity необходимо знать интенсивность, направление и цвет падающего на него света. Эти параметры задаются через расставленные в сцене объекты освещения. В данном случае используется Direction light – имитирует солнечный свет. Представляет из себя бесконечное множество параллельных друг другу лучей;

– объекты окружения. Это объекты, с которыми взаимодействует моделируемая жидкость;

– объект жидкости. Задает границы моделирования и содержит код, инициализирующий моделирование.

В зависимости от производительности устройства можно выбирать различную сложность моделирования. Она определяет радиус используемых частиц. Также при выборе низкой сложности моделирования для частицы используется меш с меньшим числом полигонов (рисунок 16).

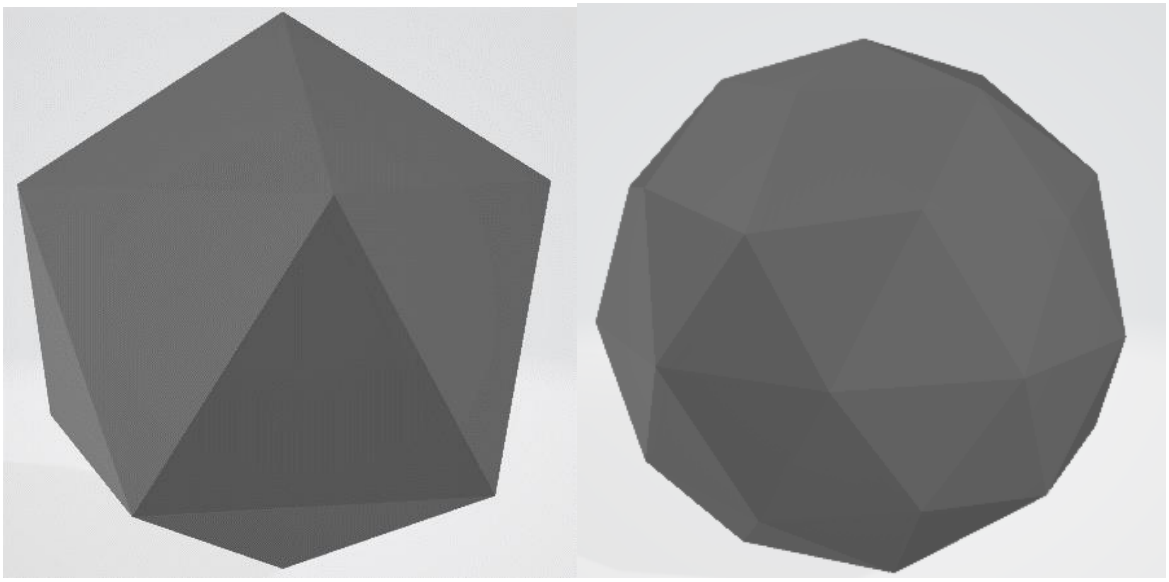


Рисунок 16 – Меш частицы

Блоки программы, отвечающие за взаимодействие с Unity и инициализацию параметров моделирования написаны на языке C#, а блоки, отвечающие за параллельные вычисления, написаны на языках HLSL и ShaderLab.

За моделирование отвечает набор классов:

Вспомогательный класс FreeCam отвечает за перемещение камеры по сцене. Он не задействован непосредственно в моделировании, но может использоваться в целях отладки. Содержит переменные для хранения скорости движения и чувствительности поворота камеры. Каждый кадр камера перемещается в зависимости от того был ли нажат какой-либо элемент управления.

BitonicSort отвечает за передачу списка частиц в вычислительный шейдер, где производится их сортировка.

Вспомогательный класс FluidBoundary, который отвечает за передачу данных о границе моделирования в вычислительные шейдеры, выполняющие расчеты движения частиц. Он хранит информацию о числе потоков, на которых будут производиться вычисления столкновений, числе граничных частиц, их радиусе и размере, плотности.

Класс FluidBody хранит параметры моделирования. Также он отвечает за инициализацию частиц (начальное положение и скорости) и передачу параметров в модуль расчета положения частиц.

Класс SmoothingKernel служит для ускорения расчетов, использующих значение ядра сглаживания. Для оптимизации вычислений в конструкторе класса заранее рассчитывается некоторые константы. Например, сохраняется значение длины сглаживания, вычисляется значение её квадрата и т. д. Также рассчитываются части уравнения ядра сглаживания, зависящие только от радиуса.

Класс GridHash предназначен для хеширования частиц в пространстве моделирования и формирования массива соседей для каждой частицы. Также класс отвечает за отображение границы ячеек на сцене в целях отладки (рисунок 20). Для этого средствами Unity на сцене проводятся линии вдоль границы пространства моделирования с шагом равным размеру ячейки.

Класс DrawLines является статическим классом, содержит лишь методы для отрисовки границ моделирования в целях отладки. Используется другими классами для отображения границ пространства моделирования, сетки хеширования, пространства начального положения частиц.

Класс ParticlesFromBounds на основе переданных данных о границах пространства моделирования, заполняет его частицами. Используется для заполнения начального объема жидкости и создания граничных частиц.

Класс FluidSolver передает данные о частицах в вычислительные шейдеры, получает результаты расчетов и обновляет параметры частиц.

Класс GridHash отвечает за построение сетки, индексирование и хеширование частиц.

Класс RenderVolume отвечает за визуализацию частиц.

Класс CBUtility отвечает за очистку буферов вычислительных шейдеров.

Класс управления FluidBodyDemo отвечает за ввод начальных параметров моделирования и управления всеми дочерними объектами. Он определяет константы для временного шага моделирования, материалов для визуализации границы моделирования, частиц, вспомогательных сеток, логические переменные для отображения отдельных частиц.

3.2 Описание работы алгоритма

Общий алгоритм работы программы представлен на рисунке 17.

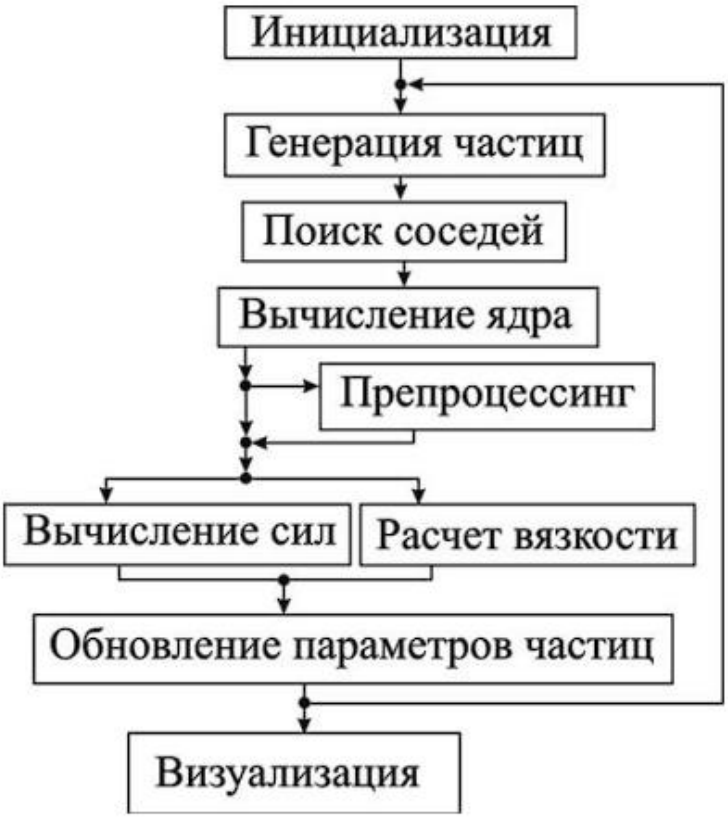


Рисунок 17 Алгоритм работы программы

Перед запуском сцены в инспекторе объекта жидкости необходимо задать параметры моделирования (Материалы для визуализации частиц, разрешение симуляции, которое определяет число используемых частиц, флаги для отображения границ моделирования и т.д.). При запуске сцены вызывается процедура класса FluidBodyDemo для инициализации моделирования. В ней из входных параметров присваиваются значения радиуса частиц и плотности.

Далее задается пространство моделирования (имеет прямоугольную форму). Его размеры вычисляется в зависимости от числа частиц. Оно состоит из внутренней и внешней границы, пространство между которыми заполняется объектами частиц (рисунки 18,19). Благодаря этому задействуется код для обработки столкновений частиц между собой, и для расчета столкновений частиц с границей не требуется писать дополнительный. На рисунке 20 представлена визуализация вспомогательной сетки поверх пространства моделирования для алгоритма поиска соседних частиц.

Для заполнения пространства между внутренней и внешней границей

используется класс ParticlesFromBounds, который хранит информацию о положении каждой частицы на границе. В его конструктор передаются объект границы моделирования, желаемое расстояние между частицами, области, которые должны остаться незаполненными (как и границы моделирования, имеют тип Bounds). На основе этих данных в цикле рассчитывается положение каждой граничной частицы и сохраняется в специальном списке класса. Также в классе сохраняется список незаполненных областей, если таковые были заданы. По умолчанию граница заполняется без пробелов с шагом равным половине радиуса частицы, таким образом не остается пустых пространств.

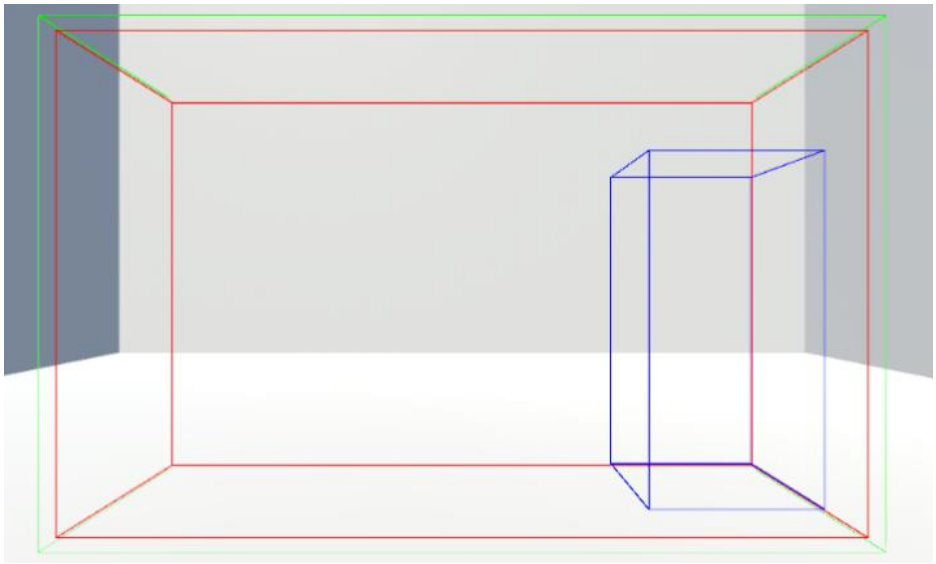


Рисунок 18 – Граница моделирования

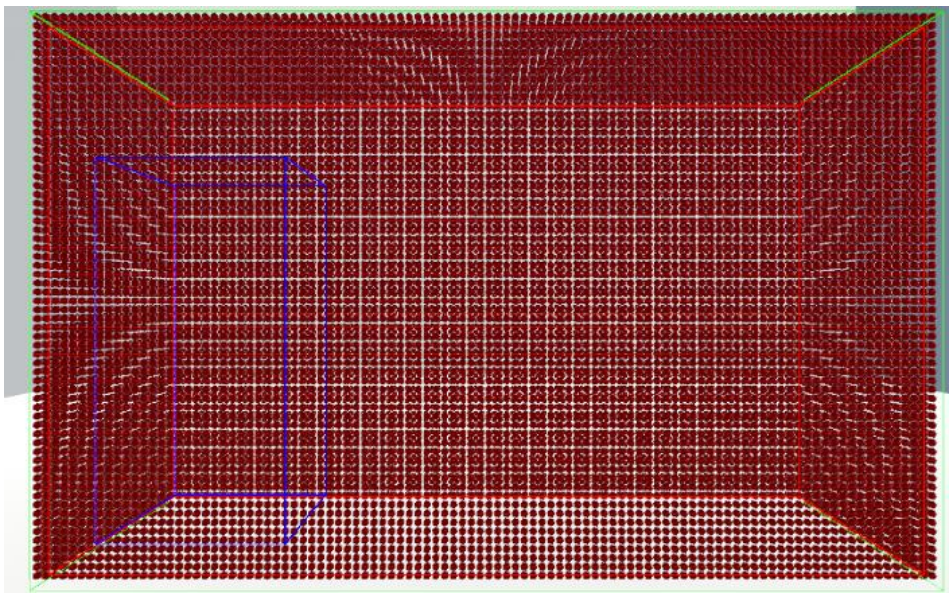


Рисунок 19 – Заполненная частицами граница моделирования

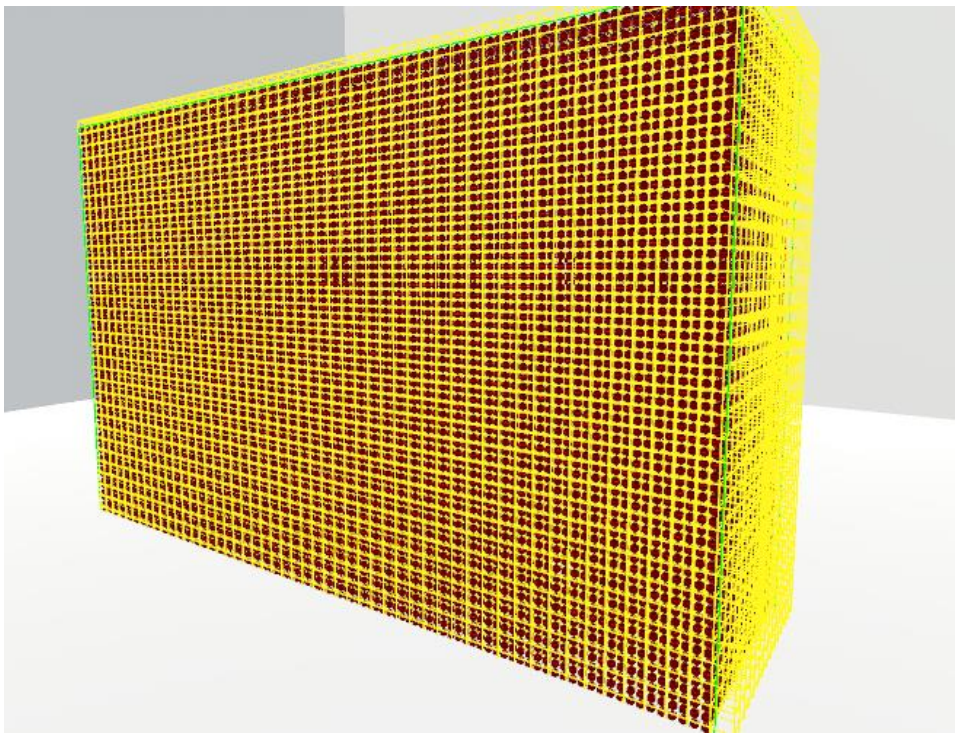


Рисунок 20 – Границы ячеек, внутри которых частицы считаются соседними

Так как с границами моделирования связано множество вычислений, для них был определена отдельная структура с типом Bounds. Она хранит всего две переменных типа трехмерный вектор – координаты центра и длина измерений границ. В ходе моделирования размеры границ могут меняться, поэтому в структуре определен не только конструкторы, создающий новый объект (в конструктор можно передать координаты центра и длины сторон, либо координаты крайних точек), но и методы для пересчета размеров границ существующего объекта, например метод для увеличения размеров границы моделирования, чтобы она включала определенную точку или область пространства. Для пересчета размеров границы в структуре определено множество вспомогательных функций, таких как возврат координат центра, размеров объекта границы, граничных точек, вычисления расстояния от точки до ближайшей границы. Также они используются для определения коллизий объектов сцены с границами моделирования (для этого проверяем чтобы точка была расположена между крайними точками границы моделирования), например при вычислении – находится ли луч света в границе моделирования, и при реализации перегрузки операторов сравнения

(считается, что объекты границы равны, если равны координаты их центров и длины сторон).

Класс FluidBoundary записывает координаты граничных частиц в буфер данных графического процессора. После этого вычисляется размер ячейки, внутри которой будет учитываться влияние соседей и инициализируется класс SmoothingKernel который представляет ядро сглаживания.

В данной реализации выбрано следующее ядро сглаживания:

$$W(r-r_j,h)=\frac{315}{64\pi h^9}\begin{cases}\left(h^2-(r-r_j)^2\right)^3 \le r \le h\\ 0\end{cases}$$
$$\nabla W(r-r_j,h)=\frac{-45}{\pi h^6}\frac{r-r_j}{\|r-r_j\|}\begin{cases}\left(h-\|r-r_j\|\right)^2, 0 \le r \le h\\ 0\end{cases} \quad (19)$$
$$\nabla^2 W(r-r_j,h)=\frac{45}{\pi h^6}\begin{cases}\left(h-\|r-r_j\|\right), 0 \le r \le h,\\ 0\end{cases}$$

поэтому заранее рассчитываются значения $\frac{315}{64\pi h^9}, \frac{-45}{\pi h^6}, \frac{45}{\pi h^6}$. Для простоты реализации на текущем этапе длина сглаживания принята равной длине четырех радиусов частицы.

Многие вычисления выполняются параллельно на графическом процессоре, поэтому класс SmoothingKernel хранит информацию о числе потоков, на которых будут выполняться расчеты (по умолчанию 128) и специальные структуры данных для передачи информации в вычислительные шейдеры:

– IndexMap типа ComputeBuffer отвечает за хранение хеша и индексов частиц (чтобы сопоставлять частицу и ее хеш).

– Groups типа int это число частиц, для которых нужно рассчитать значение одному потоку;

– Table типа ComputeBuffer содержит информацию о том какие частицы находятся в каждой ячейке;

В конструкторе определяется граница сетки типа Bounds, равная пространству моделирования, выделяются области памяти для буферов вычислительных шейдеров, инициализируется класс BitonicSort.

Для выполнения параллельных вычислений в Unity используются вычислительные шейдеры ComputeShader — это программы, которые запускаются на

графическом процессоре вне обычного конвейера рендеринга. Их программный код записывается в файлы с расширением compute и добавляется в ресурсы проекта. Однако при работе программы ComputeShader обращаются напрямую к буферам памяти. Поэтому для передачи данных в вычислительный шейдер служит встроенный класс ComputeBuffer. Нужно создать экземпляр класса, заполнить его из кода скрипта и передать в вычислительный шейдер. Также класс содержит методы, позволяющий читать информацию из буфера в массив.

Параллельное вычислений параметров частицы делится на несколько шагов:

- вычисление сил вязкости;
- вычисление плотности частиц;
- вычисление давления;
- вычисление силы поверхностного натяжения;
- синхронизация;
- вычисление изменения скорости частицы.

Вычисления проводим в специальном вычислительном шейдере, в который передаем два массива – в один будем записывать рассчитанные значения частиц, из другого считывать текущие состояния. Также передаем указатели на хеш-таблицу.

Расчет значений происходит по формулам, описанным во 2-й главе, где в множество соседей частицы i входят частицы, находящиеся в одной хеш-ячейке с i, а также в смежных с ней хеш-ячейках.

Для синхронизации используется метод барьера, останавливающий выполнение всех потоков, которые дошли до этого барьера до тех пор, пока его не достигнут все остальные.

Синхронизация используется, чтобы гарантировать, что параметры каждой частицы были рассчитаны, прежде чем использовать их для вычисления скорости.

На рисунке 21 изображена используемая схема хранения и обмена данных частиц в процессе моделирования. На этапе инициализации 0-й буфер

заполняется частицами, которые представляют собой начальные условия симуляции. На рисунке 21 этот этап обозначен в виде пунктирной стрелки, так как выполняется только в начале. Сплошными линиями показаны операции чтения из буферов и записи в буфер. В конце каждой итерации производится дополнительная запись в буфер вершин «Particles 2», который будет использоваться на этапе визуализации.

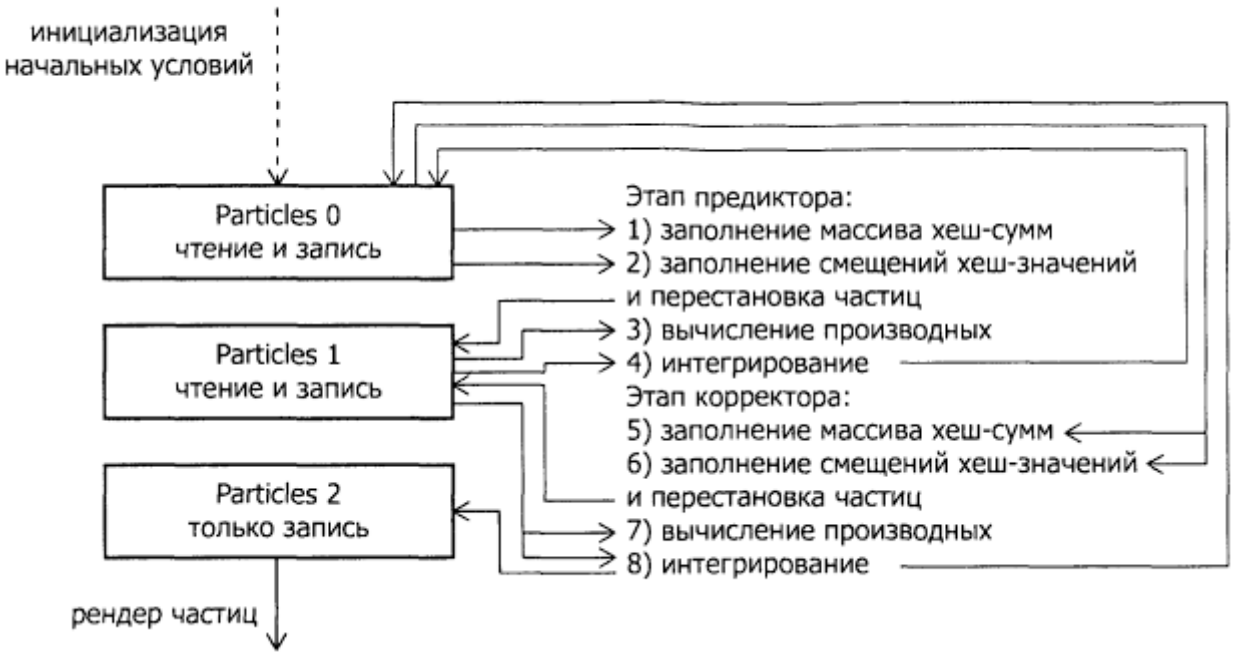


Рисунок 21 – Буферы частиц

3.3 Реализация алгоритма поиска соседей

В начале алгоритма поиска соседей структура, хранящая положение ячеек, оптимизируется. Частицы группируются по ячейкам и сортируются по индексу ячейки и позиции частицы в этой ячейке. Это позволит хранить частицы, имеющие высокую вероятность взаимодействия в соседних ячейках памяти, что позволит снизить время доступа при параллельном обращении.

Для этого сначала с помощью класса GridHash каждая частица сначала индексируются, затем вычисляется её координата относительно сетки. Этот этап называется хешированием. Значение хеш-функции позиций частиц вычисляется параллельно для каждой частицы и является результатом вычисления выражений:

$$hash(r) = h^z d^x d^y + h^y d^x + h^x,$$
$$h^a = (r_i^a - r_{\min}^a)/cell \, mod \, d^a, \quad (20)$$

где α — одна из пространственных координат (x, y или z), r_{min} — минимально возможные координаты частицы, r_i — позиция i-й частицы, cell — размер ячейки, d — целочисленный вектор, содержащий размерность сетки по каждому измерению.

Частицы с одинаковым значение hash(r) группируются последовательно.

В специальную структуру идентификатор частицы записывается как ключ, а координаты как значение. В отдельном вычислительном шейдере частицы с помощью битонной сортировки упорядочиваются по значению ячейки.

Битонная сортировка — это параллельный алгоритм сортировки, который выполняет сравнения O (n2log n). Хотя количество сравнений больше, чем в любом другом популярном алгоритме, он больше подходит для параллельной реализации, потому что элементы сравниваются в предопределенной последовательности, которая не зависит от сортируемых данных.

Исходный массив рекурсивно разбивается на две части пока в нем не останется два элемента. Далее эти элементы нужно отсортировать по возрастанию если это левая часть родительского массива и по убыванию если правая. Таким образом мы получаем битонную последовательность. После полученные пары нужно объединить. Для этого попарно сравниваем значения из левой и правой части. Если это левая часть родительского массива, то значения должны быть расположены по возрастанию, если правая, то по убыванию. Далее еще раз сортируем пары левой и правой части. На следующей итерации используются пары родительского массива. Принцип работы алгоритма представлен на рисунке 22.

Сортировка массива частиц на каждой итерации алгоритма приводит к затратам времени, но, учитывая необходимость поиска соседних частиц для каждой частицы, механизм хэширования снижает трудоемкость поиска соседей до $O(3^s c_n N)$, где c_n — среднее количество частиц в одной ячейке, s —размерность пространства.

В результате сортировки получаем структуру, которая хранит список

частиц, упорядоченных по номеру ячейки. Каждая строчка этой структуры индексируется. Также для каждой ячейки определяем начальный и конечный индекс хранения частиц. Хотя эти данные можно использовать для перебора соседей в каждом потоке, в этом случае придется для каждого соседа проверять расстояние до исходной частицы. Чтобы избежать лишних вычислений и обращений к памяти, соседи для каждой частицы рассчитываются в начале каждого шага моделирования после формирования структуры значений хеш-функции. На этом же этапе проверяется расстояние до частицы и адрес соседа записывается в память GPU. Таким образом в ячейку отсортированной структуры уже попадают соседние частицы.

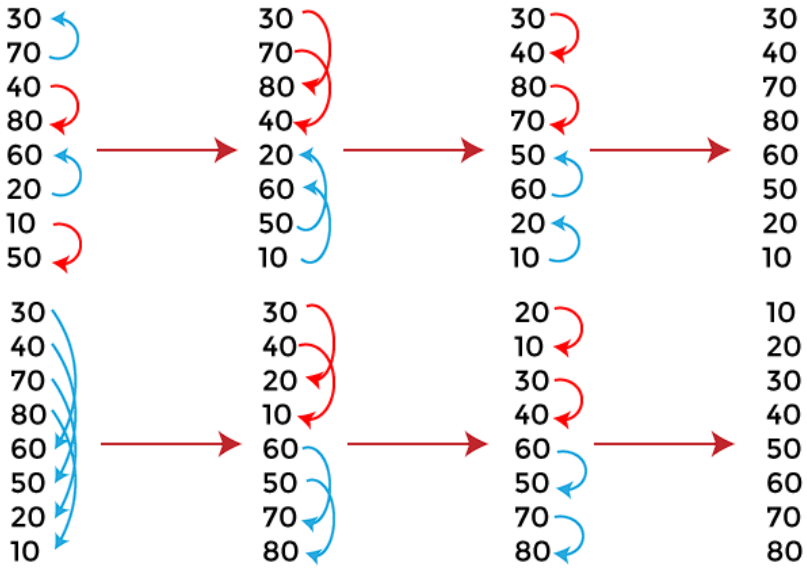
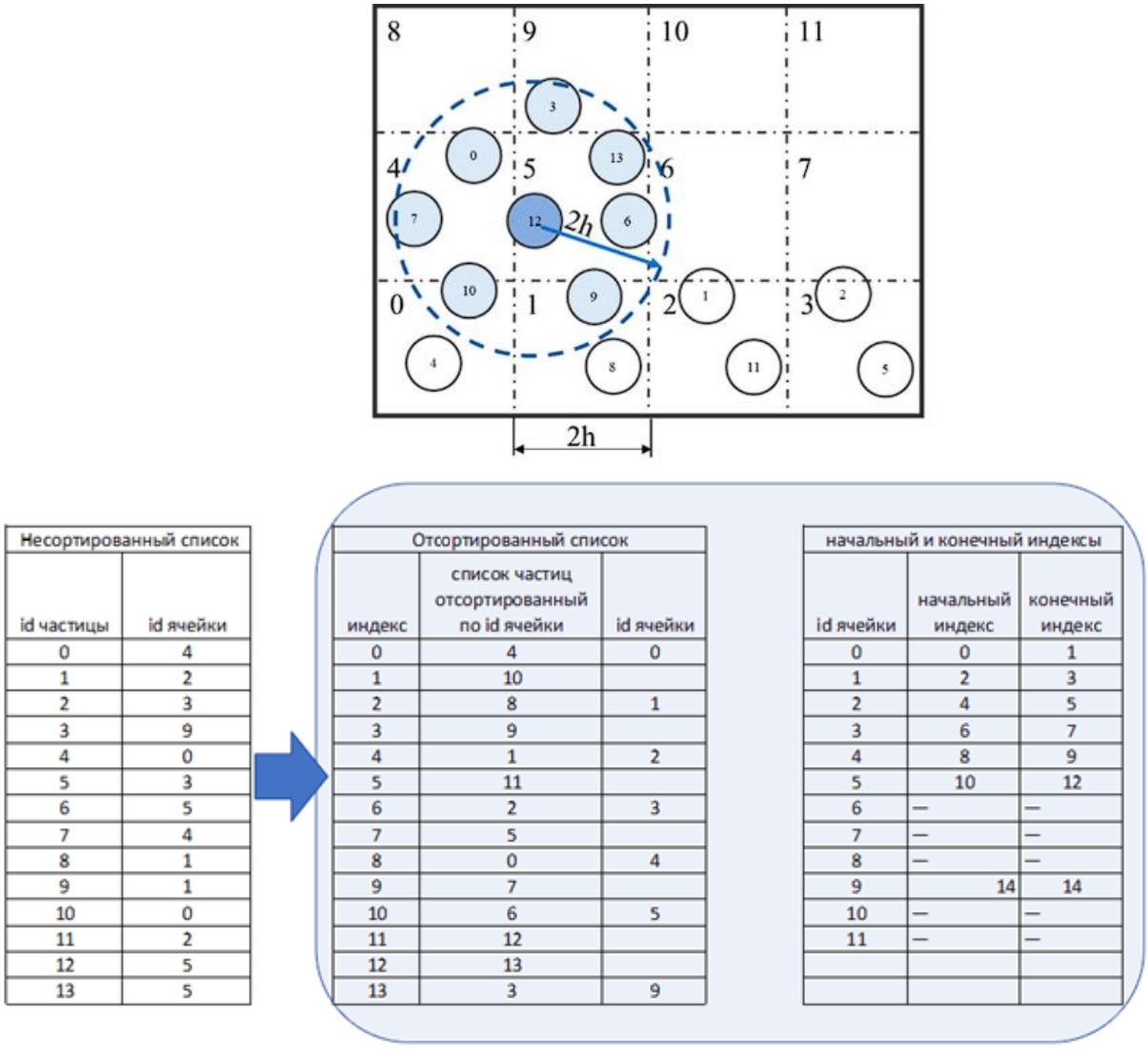


Рисунок 22 – Принцип работы битонной сортировки

Требование пересчитывать соседей связано с тем, что частицы постоянно перемещаются, и множество соседей будет отличаться на различных шагах моделирования. Соседей можно пересчитывать немного реже, например 1 раз в 3–4 последовательных шага моделирования, но это не дает значительных преимуществ в скорости вычислений, так как на этапе итерационного вычисления давления возрастают ошибки.

В дальнейшем при вычислении значений частицы поток считает начальный и конечный индекс для ячейки и определит область памяти, в которой находятся соседние ячейки. Так как структура отсортирована – данные о координатах

будут находиться в соседних ячейках памяти, что увеличит скорость доступа. Общий алгоритм сортировки представлен на рисунке 23.



Несортированный список

id частицы	id ячейки
0	4
1	2
2	3
3	9
4	0
5	3
6	5
7	4
8	1
9	1
10	0
11	2
12	5
13	5

Отсортированный список

индекс	список частиц отсортированный по id ячейки	id ячейки
0	4	0
1	10	
2	8	1
3	9	
4	1	2
5	11	
6	2	3
7	5	
8	0	4
9	7	
10	6	5
11	12	
12	13	
13	3	9

начальный и конечный индексы

id ячейки	начальный индекс	конечный индекс
0	0	1
1	2	3
2	4	5
3	6	7
4	8	9
5	10	12
6	—	—
7	—	—
8	—	—
9	14	14
10	—	—
11	—	—

Рисунок 23 – Принцип хеширования частиц

При реализации поиска соседних частиц следует учитывать особенности распределения памяти на GPU – память выделяется для всех переменных заранее, до начала вычислительного процесса. Отсутствие возможности динамического выделения памяти для массива соседних частиц указывает на необходимость предварительной оценки размера данного массива.

Приблизительная величина массива соседей вычисляется как отношение объема области сглаживания частицы к объему частицы (такая модель адекватна только при слабой сжимаемости жидкости). Но при резком сжатии, особенно при ударах, число соседей может значительно возрасти, поэтому требуется выделить

дополнительную память, как минимум для хранения 50–60 адресов соседей.

3.4 Визуализация частиц

Как уже отмечалось выше обработка данных графического конвейера возможна на этапах вершинной и фрагментарной программ. Для визуализации частиц жидкости используется метод создания изображения в экранном пространстве. Данный метод является бессеточным, что означает что треугольная сетка не строится в процессе получения изображения. Вместо этого в буфере кадра создается снимок исходных данных и на следующих шагах изображение обрабатывается в спроецированном пространстве.

Для данного метода используется следующий алгоритм отображения:

- отображение точек в виде сфер;
- сглаживание карты глубины;
- применение текстуры кругового градиента и цвета.

В качестве входных данных алгоритм использует данные буфера кадра (цвета и глубины), полученные при рендере непрозрачных объектов, находящихся на сцене. Буфер цвета и круговой градиент используются для имитирования полупрозрачности и преломления света. Буфер глубины сцены используется для определения, какой объект (частица или объект сцены) для данного фрагмента находится ближе к наблюдателю. Это нужно для корректного отображения перекрытия объектов и жидкости. Также при моделировании жидкостей методом сглаженных частиц мы имеем множество точек и длину сглаживания, одинаковую для всех частиц.

Современные графические адаптеры позволяют отображать достаточно большое количество геометрических точек, однако для экономии ресурсов вместо рендера тысяч полигональных сфер будем выводить их с помощью точечных спрайтов. Для точечных спрайтов необходимо рассчитать их размер на экране, попиксельные изменения в буфере глубины и попиксельное значение толщины слоя жидкости.

Поверхность частицы визуализируется с помощью поверхностных шейдеров. Поверхностные шейдеры используются если предполагается, что объект

будет взаимодействовать с источником света и тенями. Для того чтобы применить шейдер к объекту на сцене требуется использовать материал. Это экземпляр шейдера с параметрами.

Для реализации создадим в проекте поверхностный шейдер и материал для визуализации частиц (рисунок 24). Данный материал будет применять текстуру, сгенерированную шейдером к каждой частице в кадре.

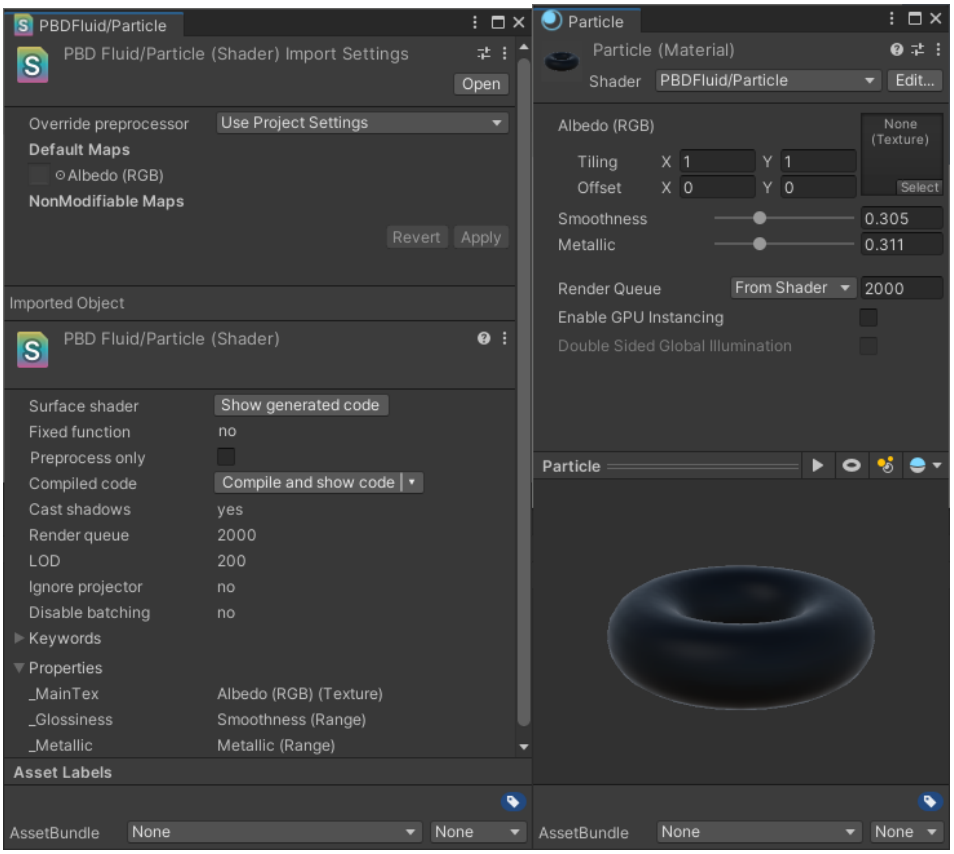


Рисунок 24 – Шейдер и материал для визуализации частиц

Структура файла поверхностного шейдера такова: вверху файла указывается название и путь произвольной вложенности. Далее указывается блок с настройками. Это связующее звено между редактором и кодом шейдера. Сначала указывается имя параметра, которое будет использоваться в коде шейдера. Затем имя параметра, которое будет отображаться в инспекторе материала в Unity. Далее тип параметра, от которого зависит вид поля для ввода параметра, например цвет, текстура, вектор, число, ограниченное неким диапазоном и т. д. Настройки указываются на языке ShaderLab, что также влияет на используемые типы данных, которые будут отличаться от типов, указанных в блоке SubShader.

В шейдере, отвечающем за поверхность частицы определены следующие параметры:

– _Color – цвет в Unity представлен в формате RGB;

– _MainTex – главная текстура, имеет встроенный тип 2D.В качестве текстуры используется белый цвет;

В блоке CGPROGRAM указаны следующие директивы:

– для использования встроенной в Unity стандартной модели освещения, с генерацией тени от любого источника света (по умолчанию шейдеры поддерживают только тени от однонаправленного источника света при прямом рендеринге);

– для поддержки создания экземпляров графического процессора. Это метод оптимизации вызовов отрисовки, который отображает несколько копий меша с одним и тем же материалом за один вызов отрисовки. Каждая копия меша называется экземпляром. Это хорошо подходит для визуализации множества одинаковых частиц;

– директива для указания функции настройки вершин. Она загружает позиционные данные для каждого экземпляра частицы.

Далее идет объявление переменных, с которыми шейдер работает. Они соответствуют параметрам, указанным в блоке Properties:

– sampler2D _MainTex – хранит текстуру частицы;

– float4 color хранит цвет частицы;

– float diameter хранит диаметр частицы;

В структуре входных данных input для главной поверхностной функции Surf указан следующие параметры:

– float2 uv_MainTex это двухкомпонентный вектор, по которому считывается цвет пикселя текстуры;

– float4 color это четырёхкомпонентный вектор, который хранит значение основного цвета частицы.

Функция Surf запускается для каждого пикселя, занимаемого объектом. Она принимает в качестве параметров текстуру с uv-координатами и

результирующую текстуру, которая затем будет использоваться Unity для отображения.

В блоке свойств укажем переменную для контроля уровня прозрачности _TransVal ("Transparency Value", Range (0,1)) = 0.5.

Алгоритм, описанный в шейдере, работает следующим образом:

Размер точки, то есть сторона квадрата, полученного на выходе из вершинного шейдера, вычисляется по формуле

$$p_s = \frac{2hH}{|x_p - C|}, \tag{21}$$

где h – длина сглаживания, x_p – позиция частицы, C – позиция камеры, H – высота буфера кадра в пикселях.

Во фрагментарном шейдере сначала рассчитаем вектор нормали к сфере. Для каждого фрагмента в шейдере мы получаем локальные координаты спрайта от (0; 0) (левый нижний угол) до (1; 1) (правый верхний угол).

Так как проецируя сферу на квадратный спрайт, мы получим окружность, вписанную в квадрат, то координаты x и y вектора нормали поверхности сферы вычисляются как $x = 2x_{sprite} - 1$, где x_{sprite} – значение локальной координаты. Длина вектора нормали равна 1, поэтому координату z получим как $\sqrt{1 - N_x^2 - N_y^2}$. Аналогичным образом вычисляется значение толщины сферы.

Если $N_x^2 + N_y^2 > 1$, то данный фрагмент находится за пределами проекции сферы и его необходимо отбросить.

Для расчета значения глубины сферы сначала в вершинном шейдере вычислим позицию вершины в пространстве вида, то есть умножим матрицу вида на позицию вершины x^{world} в мировом пространстве, дополненной четвертым компонентом w=1:

$$x^{eye} = M_{view}^T \{x_x^{world}, x_y^{world}, x_z^{world}, 1\}^T \tag{22}$$

Матрица вида – матрица, преобразующая мировые координаты в пространства просмотра. Матрица проекций отвечает за то, как вершины графических примитивов буду проецироваться на экран монитора. Воздействуя на матрицу

проекций, можно добиться масштабирования, смещения, вращения результирующего изображения.

В данном случае матрица вида содержит только поворот и смещение в трехмерном пространстве, поэтому у результирующего четырехмерного вектора можно отбросить компоненту w и считать вектор x_{eye} трехмерным. Во фрагментарном шейдере к вектору добавим вектор $N \cdot r_p$ и трансформируем полученный вектор матрицей проекции:

$$x_x^{pixel} = x_x^{eye} + N_x \cdot r_p$$
$$x_y^{pixel} = x_y^{eye} + N_y \cdot r_p$$
$$x_z^{pixel} = x_z^{eye} + N_z \cdot r_p$$
$$x^{clip} = M_{projection}^T \cdot \left(x^{pixel}\right)^T.$$

Значение глубины фрагмента определяется как $depth = \frac{x_z^{clip}}{x_w^{clip}} \cdot 0.5 + 0.5$.

Таким образом, из массива частиц можно получить текстуры глубины и толщины, которые по размерам совпадают с областью вывода изображения. Далее эти текстуры будут использоваться для построения поверхности жидкости в плоскости экрана.

Затем необходимо рассчитать нормали частиц. Это делается с помощью встроенного в Unity метода Recalculate.

Финальный цвет фрагмента определяется преломлением светового луча. Это позволит обозначить участки с большой плотностью частиц более темным цветом. Для вычисления цвета преломления используется двумерная текстура, содержащая цветовой буфер кадра. Значение из этой текстуры берется по координатам $tC_{reefact} = tC + N_{xy} \cdot (\gamma \cdot thickness)$, где tC – текстурные координаты текущего фрагмента, N_{xy} – вектор, состоящий из компонент x и y вектора нормали, γ – коэффициент, определяющий силу преломления, thickness – толщина слоя жидкости в данной точке. Преломленный компонент цвета вычисляется как линейная интерполяция между цветом самой жидкости и цветом буфера кадра по координатам tC_{refact} с коэффициентом $exp^{-thickness}$. Полученное значение цвета

умножается на коэффициент Френеля с параметром $f = N \cdot V$, где V – нормализованный вектор вида.

Коэффициент Френеля приближенно вычисляется по формуле:

$$k^{Fresnel}(f) = R_0 + (1 - R_0)(1 - f)^5 \quad (23)$$

где R_0 – константа, в данном случае равная 0.2.

Так как жидкость наблюдателю представляется как гладкая и сплошная среда, то для повышения реалистичности отображения необходимо сгладить границы между сферами. Используем для этого текстуру кругового градиента, которую наложим на отображаемые сферы. Для реализации создадим в проекте C# скрипт ProceduralTexture и назначим его пустому объекту GameObjcet. В скрипте нужно разместить код, который создаст круговой градиент, отобразит его на текстуре и передаст в шейдер.

В скрипте добавим переменные для изменения высоты и ширины текстуры, переменную типа Texture2D для хранения сгенерированной текстуры, а также переменные для хранения материала и двумерный вектор для хранения координат центра.

В начале работы скрипта проверяем, есть ли материал у объекта. Если есть – вызовем функцию для создания градиента и результат сохраним в созданную переменную, иначе выведем предупреждение. Сгенерированную текстуру затем присвоим материалу объекта.

Алгоритм создания кругового градиента работает по следующему принципу: сначала создается новая текстура и в ранее созданный двумерный вектор запишем центр текстуры. Далее в цикле для каждого пиксела рассчитаем расстояние от центра и в зависимости от этого присвоим значения цвета. Запишем все изменения в текстуру и вернем её в основную программу шедйера.

В шейдере также нужно написать логику для обработки созданной текстуры градиента для отображения эффекта прозрачности. Чтобы шейдер применял эффект прозрачности нужно в секции #pragma использовать параметр alpha.

В функции surf добавим код, который будет присваивать каждому пикселю результирующей поверхности значение прозрачности в зависимости от цвета

входящей текстуры. Черному цвету текстуры соответствует полная прозрачность, белому наоборот (рисунок 25).

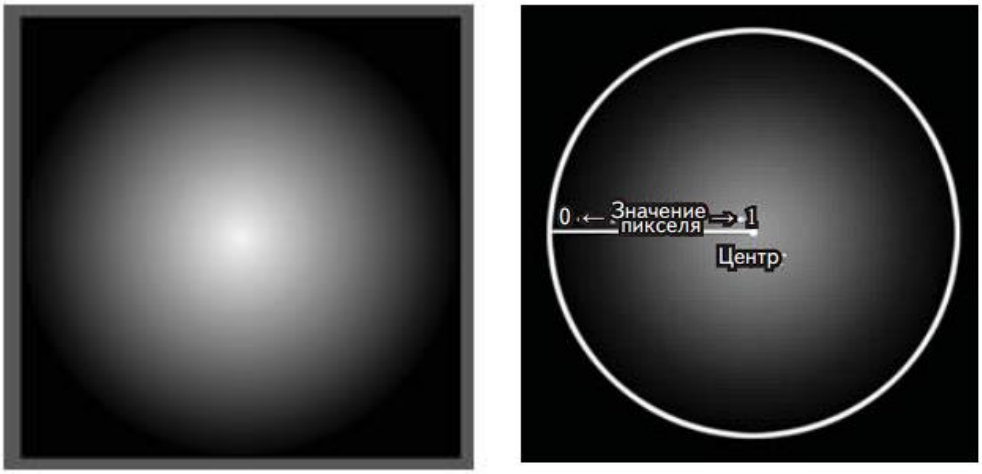


Рисунок 25 – Результирующая текстура кругового градиента

3.5 Результаты фактического тестирования

Для тестирования программного продукта было подготовлен несколько сцен, демонстрирующих разные сценарии применения разработанной системы. Так как вычислительные мощности мобильных устройств на сегодняшний день недостаточно велики для физически точных расчетов движения огромного числа частиц (>100 000) в режиме реального времени, в ходе реализации алгоритма был сделан ряд упрощений с целью оптимизации вычислений, поэтому результаты моделирования сложно сравнивать с экспериментальными данными.

Основное предназначение разработки – создание художественных эффектов, поэтому основное внимание при тестировании будет уделено визуальной достоверности моделирования и производительности.

Тестирование велось на устройстве Samsung Galaxy A34 со следующими характеристиками:

- операционная система Android 13;
- процессор Dimensity 1080, 2x Cortex-A78 2.6 ГГц, 6x Cortex-A55 2 ГГц;
- графический ускоритель Mali-G68 MC4;
- объем оперативной памяти 8 Гб.

Данные получены с помощью удаленного профайлинга через WiFi для

устройств Android.

Первая сцена моделирует взрыв внутри объема жидкости. На рисунке 26 пространство моделирования ограничено кубом, внутри которого создается объем жидкости. В данной сцене используется 25 000 частиц. На рисунке видно, что изначально весь объем тела жидкости был равномерно заполнен частицами, следовательно алгоритм инициализации выполнился корректно.

Также с целью тестирования на сцену добавлен специальный объект, который после начала моделирования задаст начальную скорость всем частицам в определенном радиусе. Центр объекта находится внутри пространства жидкости. Таким образом имитируется эффект взрыва. На рисунке 27 представлены результаты моделирования. Можно сказать, что с визуальной точки зрения они получились достаточно достоверными, чтобы использоваться в качестве художественного эффекта.

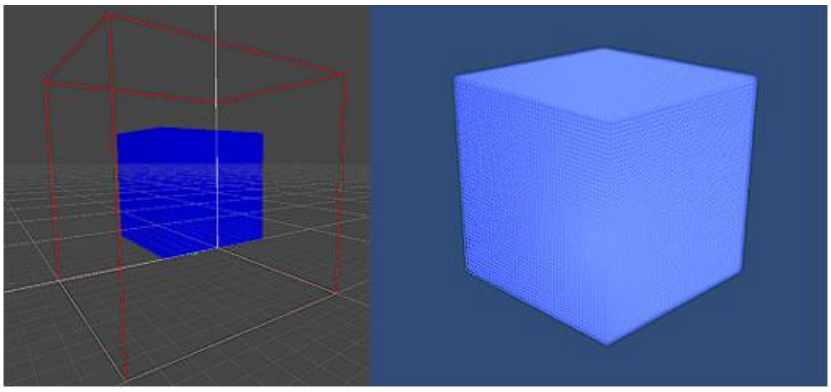


Рисунок 26 – Изначально созданный объем жидкости

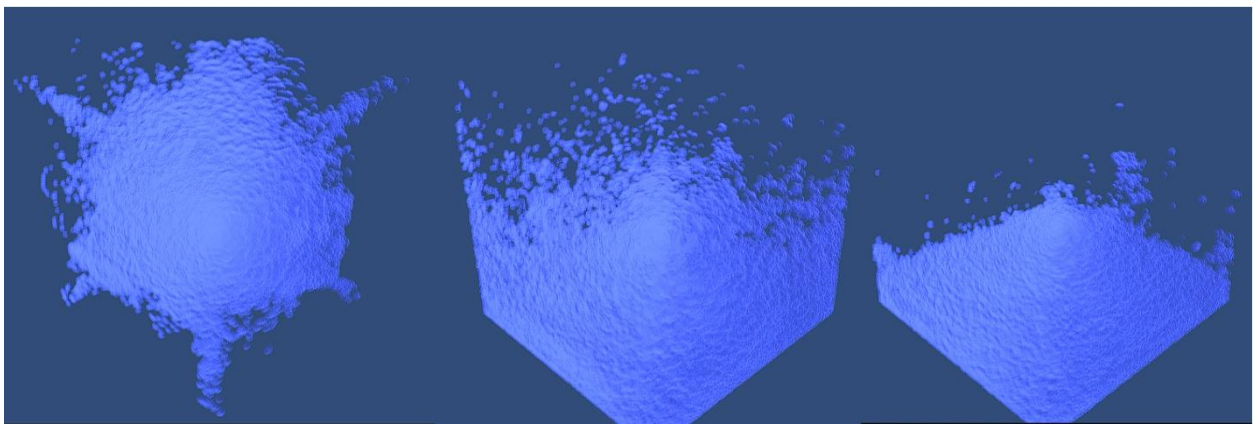


Рисунок 27 – Демонстрация моделирования взрыва

Также стоит обратить внимание на производительность. На рисунке 28 представлены данные об использовании приложением оперативной памяти, собранные с помощью встроенных в Unity инструментов профилирования за все время моделирования.

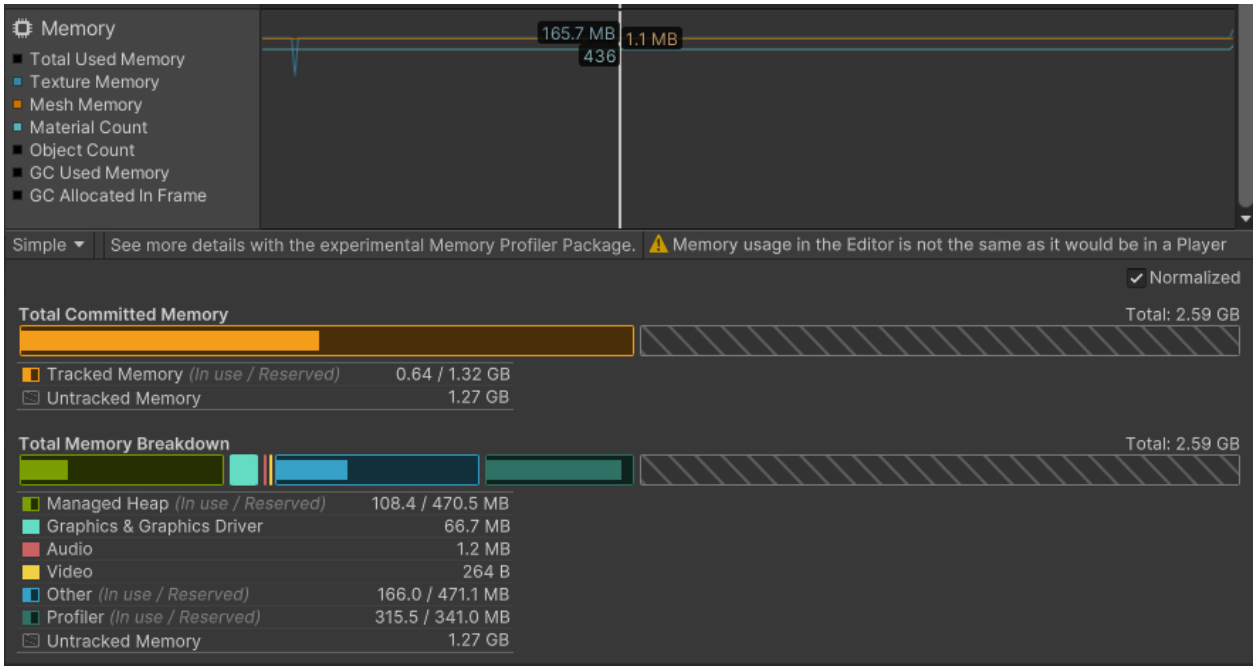


Рисунок 28 – Статистика использования оперативной памяти

На рисунке видно, что из отслеживаемого объёма памяти приложением используется всего 300 мб (без учета памяти, занимаемой инструментом профилирования), что вполне допустимо для современных мобильных устройств. При этом большая часть памяти задействуется для хранения текстур частиц. Также на графике использования видно, что память была выделена при начале моделирования, и далее ее объем не менялся. Значит для системы можно предсказать объем потребляемой памяти и исходя из этого настраивать различные параметры моделирования, тем самым влияя на достоверность и детализированность поведения жидкости, в зависимости от производительности целевого устройства.

Данные о нагрузке на графический процессор представлены на рисунке 29. На графике видно, что средняя частота кадров все время моделирования составляла около 100 кадров в секунду, что является достаточно хорошим показателем и обеспечивает плавное движение картинки. При этом нагрузка была стабильной, просадок в производительности не наблюдалось.

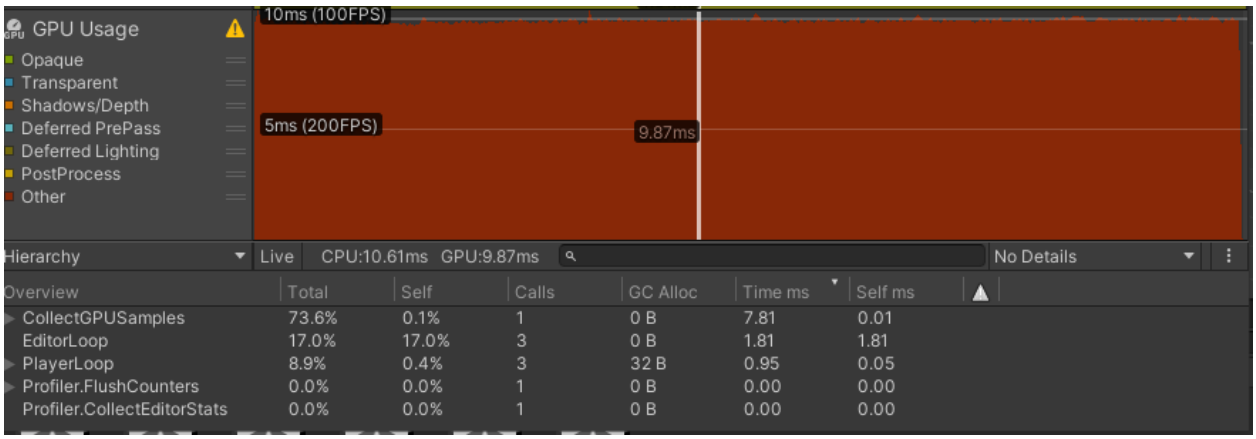


Рисунок 29 – Статистика нагрузки на графический процессор

То же самое видно на графике нагрузки на центральный процессор (рисунок 30). В целом нагрузка была достаточно стабильной, просадка производительности произошла лишь в момент активизации сборщика мусора, и при этом не была критичной. Совокупное время, затраченное на выполнение программных скриптов, рендеринг графики на процессоре, сборку мусора составляет лишь 1 миллисекунду. Поэтому можно сделать вывод что быстродействие программы упирается в мощность графического процессора и в среднем при моделировании этой сцены частота кадров составит около 100 кадров в секунду.

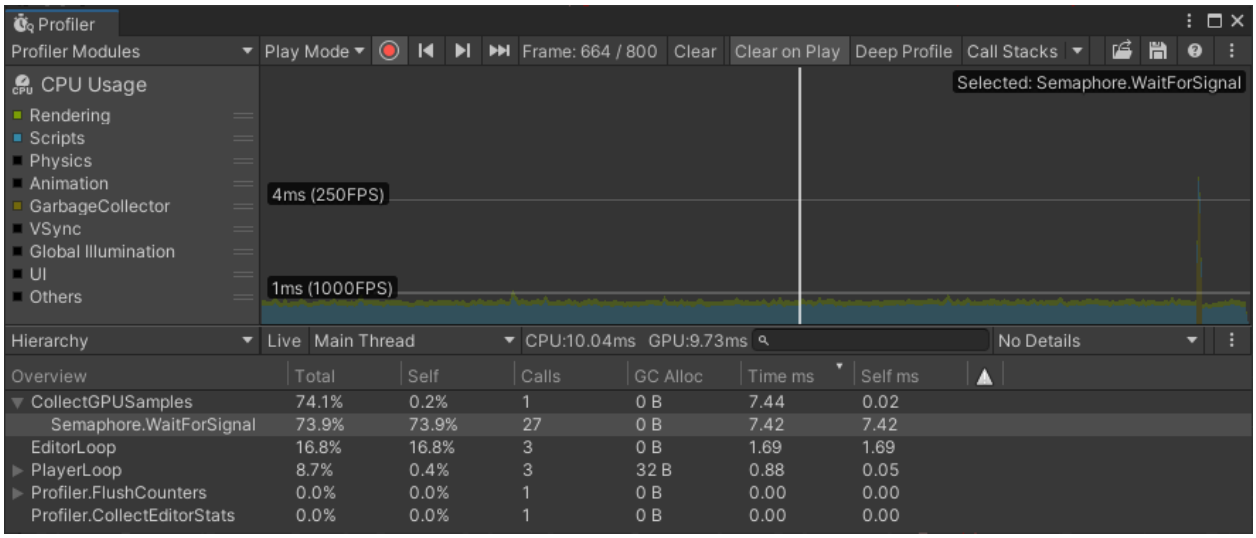


Рисунок 30 – Статистика нагрузки на центральный процессор

Запустим еще одну тестовую сцену, где на пути потока жидкости расставлены препятствия. Её исходный вид показан на рисунке 31. На сцене так же, как и в первой тестовой сцене присутствует объект, который при начале

моделирования задаст частицам начальную скорость, заставив их двигаться по направлению к препятствиям. Например, таким образом можно имитировать движение волн. Система должна регистрировать столкновения частиц с мешем препятствия и применять к ним граничные условия.

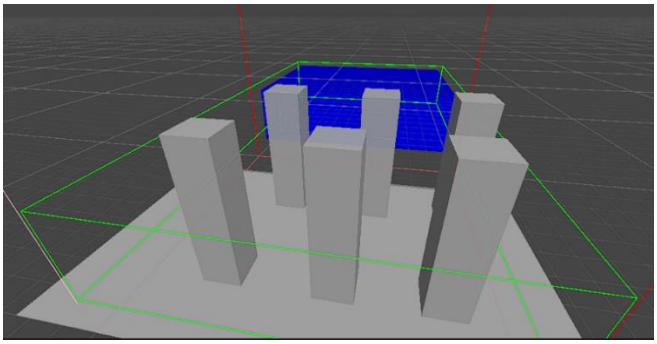


Рисунок 31 – Исходный вид сцены

На рисунке 32 представлены результаты моделирования. На рисунке видно, как волны огибают препятствия, поэтому можно сделать вывод что алгоритм обработки столкновений работает корректно. Данный результат так же выглядит достаточно реалистично для создания художественного эффекта.

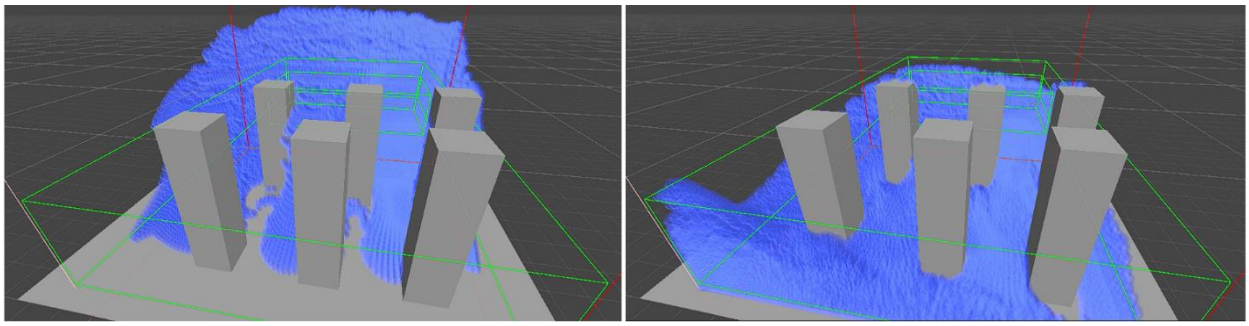


Рисунок 32 – Результаты моделирования движения волны

На рисунке 33 представлены данные профайлера о затрачиваемой оперативной памяти. В этой сцене объем также составил около 360 мб, что также приемлемо для современных мобильных устройств. Объем потребляемой памяти в этой сцене также не менялся.

Нагрузка на графический процессор оказалась стабильной (рисунок 34). Средняя частота кадров составила 120.

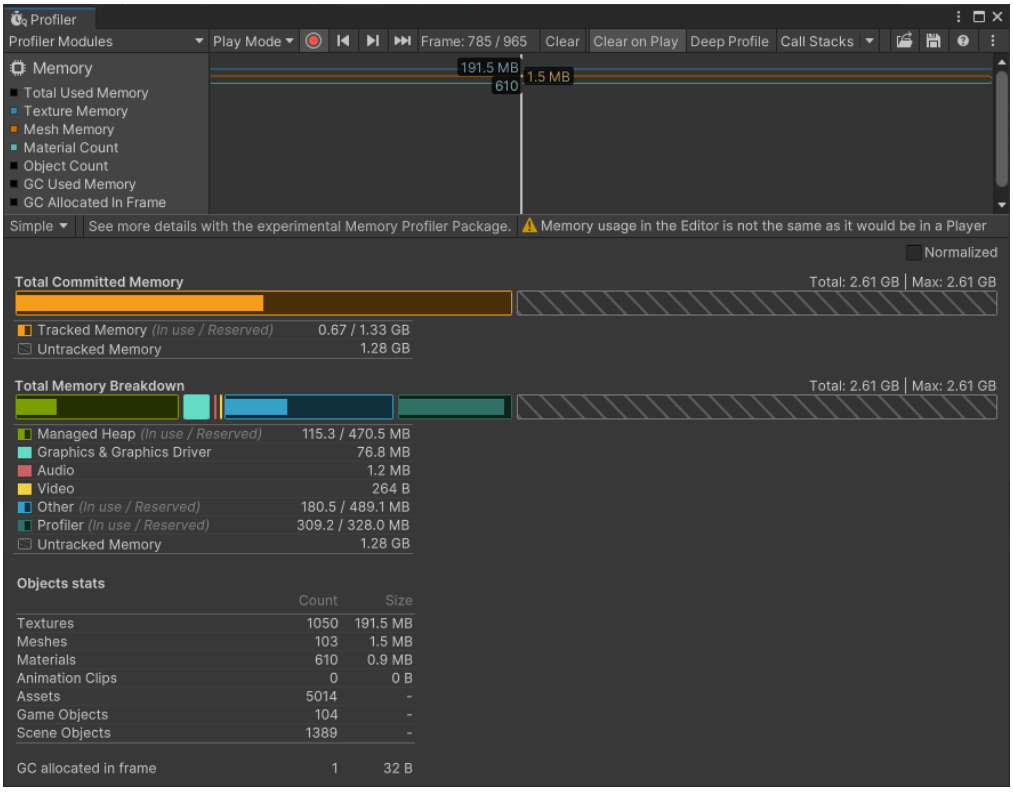


Рисунок 33 –Нагрузка на оперативную память при движении волны

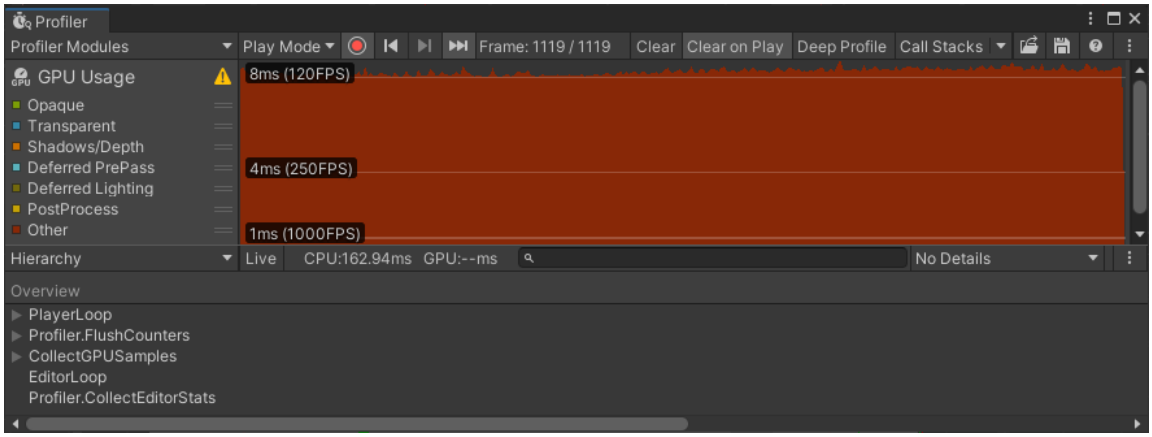


Рисунок 34 – Нагрузка на GPU при движении волны

Нагрузка на процессор была постоянной (рисунок 35). На рендеринг средствами ЦП и выполнение программных скриптов в среднем затрачивалось не более 1 миллисекунды, поэтому производительность снова упирается в мощность графического процессора.

В целом анализ производительности показал, что разработанная система достаточно оптимизирована и вполне может использоваться в проектах, предназначенных для мобильных устройств.

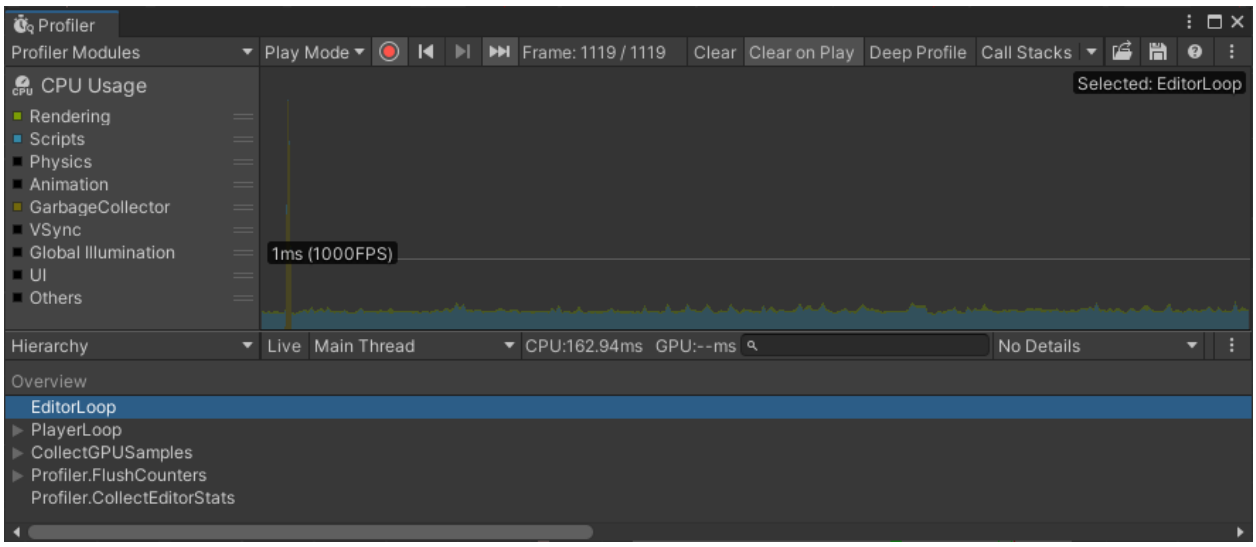


Рисунок 35 – Нагрузка на ЦП при движении волны

3.6 Анализ практической значимости результатов

Жидкости, в частности вода, ответственны за многие визуально богатые явления, и их моделирование давно вызывает интерес и вызов компьютерной графике. Во многих играх, из-за большого объема вычислений, симуляция жидкостей ограничивается созданием визуальных эффектов, шейдеров, анимаций, которые не могут использоваться в игровых механиках. Но с ростом вычислительных мощностей стало возможным создать систему симуляции жидкости, предоставляющую возможности для взаимодействия с игроком. Использование такой системы потенциально может привести к появлению новых игровых механик и новых подходов к разработке видеоигр.

Цель состояла в том, чтобы создать симуляцию несжимаемой жидкости для среды Unity, способную взаимодействовать с твердыми телами и при этом работать со скоростью не менее 30 кадров в секунду. С этой целью был реализован алгоритм численного решения уравнения Навье-Стокса с рядом оптимизаций, специально предназначенный для использования в мобильных видеоиграх, работающий достаточно быстро и стабильно.

В результате задача моделирования движения несжимаемого потока жидкости в реальном времени в среде Unity был решена с использованием метода сглаженных частиц. В ходе работы был получен набор объектов и компонентов с

настраиваемыми параметрами моделирования (таких как воздействие гравитации, вязкость) для создания объема жидкости, пространства моделирования, интерактивных объектов, способных взаимодействовать с жидкостью. Создан набор графический шейдеров, позволяющих визуализировать жидкость. Разработано тестовое приложение, содержащее набор сцен, демонстрирующих возможности применения системы симуляции.

Разработанная система достаточно оптимизирована, чтобы использоваться в проектах для мобильных устройств и может применяться разработчиками видеоигр для создания художественных эффектов и игровых механик.

ЗАКЛЮЧЕНИЕ

В ходе выполнения индивидуального задания сначала была проанализирована предметная область. Для этого было рассмотрено понятие программного средства для симуляции физических явлений. Далее описаны методы оптимизации мобильных видеоигр. Рассмотрены различные подходы к моделированию жидкостей в сфере разработки видеоигр, сформулированы требования к методу моделирования. В итоге был выбран наиболее подходящий для данной предметной области метод SPH. Также был проведен обзор существующих решений для симуляции движения жидкости и выполнен их сравнительный анализ. Обоснована необходимость разработки нового программного продукта и сформулированы предварительные требования к его функционалу.

На этапе алгоритмического решения задачи была описана математическая модель моделирования. Затем описаны подходы к решению задачи «поиска соседей» в рамках метода SPH. Выделены особенности визуализации частиц в среде разработки Unity. Обоснован выбор программных средств для разработки и дан обзор возможностей профильного программного обеспечения.

Выполнено проектирование программы, в результате которого было получено описание используемых классов, диаграмма взаимодействия функциональных модулей, построен алгоритм работы программы, определены входные данные. Далее в среде Unity была разработана система для моделирования движения жидкости с помощью метода SPH, способная работать на мобильных устройствах.

Выполнено тестирование разработанного программного продукта, проанализирована практическая значимость полученных результатов.

В результате получена система симуляции жидкости для среды разработки Unity с настраиваемыми параметрами моделирования, поддерживающая взаимодействие с другими объектами в сцене. Данная система может быть с легкостью внедрена в сторонние Unity проекты и использована для создания визуальных эффектов или в игровых механиках.

БИБЛИОГРАФИЧЕСКИЕ ССЫЛКИ

1 Общая характеристика игровых движков [Электронный ресурс]. – URL:https://studbooks.net/2039980/informatika/obschaya_harakteristika_igrovyh_dvizhkov (дата обращения: 30.05.2023).

2 ИСПОЛЬЗОВАНИЕ ПРОГРАММНОГО ПРИЛОЖЕНИЯ UNITY ДЛЯ 3D-МОДЕЛИРОВАНИЯ ФИЗИЧЕСКИХ ОБЪЕКТОВ [Электронный ресурс]. – URL: https://elibrary.ru/item.asp?id=49395407 (дата обращения: 30.05.2023).

3 Физический движок / Текст : электронный // Википедия / Page Version ID: 121583295. – 2022. – URL: https://ru.wikipedia.org/w/index.php?title=%D0%A4%D0%B8%D0%B7%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B4%D0%B2%D0%B8%D0%B6%D0%BE%D0%BA&oldid=121583295 (дата обращения: 30.05.2023).

4 Technologies U. Unity - Manual: Physics [Электронный ресурс]. – URL: https://docs.unity3d.com/Manual/PhysicsSection.html (дата обращения: 30.05.2023).

5 Шарыгин А.Ю. Основные Процессы Оптимизации Мобильной Игры На Unity / А.Ю. Шарыгин, С.П. Кайгордов, С.П. Кайгородов // Аллея Науки. – 2018. – Т. 5. – № 5 (21).

6 Сиротина И.К. Методы Оптимизации Графических Файлов Видеоигр / И.К. Сиротина, А.А. Мышковский, К.Э. Остапчук // Актуальные Научные Исследования В Современном Мире. – 2021. – № 2- – 2 (70).

7 Там же. С. 146

8 Technologies U. Unity - Manual: Compressing mesh data [Электронный ресурс]. – URL: https://docs.unity3d.com/Manual/mesh-compression.html (дата обращения: 05.06.2023).

9 Fluid-flow modeling and stability analysis of communication networks : 20th IFAC World Congress / N. Espitia [et al.] // IFAC-PapersOnLine. – 2017. – Vol. 50. – № 1. – P. 4534-4539.

10 Субстанциальная производная и ее использование в гидравлике / О.И.

Зайцев [и др.] // Строительство Уникальных Зданий И Сооружений. – 2013. – № 8 (13).

11 Евстигнеев Н.М. Высокоскоростные Параллельные Алгоритмы Решения Задач Механики Сплошной Среды Методом Сглаженных Частиц / Н.М. Евстигнеев, Ф.С. Зайцев, О.И. Рябков // Доклады Академии Наук. – 2014. – Т. 459. – № 3.

12 Иванович Н.Е. Моделирование течений слабосжимаемой жидкости методом сглаженных частиц на графических процессорах / Н.Е. Иванович // Известия Самарского научного центра Российской академии наук. – 2016. – Т. 18. – № 2-3. – С. 936-940.

13 Суравикин А.Ю. Моделирование и визуализация несжимаемых жидкостей методом сглаженных частиц на графическом процессоре : кандидат технических наук / А.Ю. Суравикин. – Омск, 2012.

14 Giraldo G. CFD Modeling, Analysis & CFD Simulation For Beginners [Электронный ресурс]. – URL: https://www.simscale.com/blog/cfd-analysis-for-beginners/ (дата обращения: 30.05.2023).

15 Храпов С.С. Численное моделирование гидродинамических аварий: размыв дамб и затопление территорий / С.С. Храпов // Вестник Санкт-Петербургского университета. Математика. Механика. Астрономия. – 2023. – Т. 10. – Численное моделирование гидродинамических аварий. – № 2. – С. 357-373.

16 Saftly W. Hierarchical octree and k-d tree grids for 3D radiative transfer simulations / W. Saftly, M. Baes, P. Camps // Astronomy & Astrophysics. – 2014. – Vol. 561. – P. A77.

17 Fast Octree Neighborhood Search for SPH Simulations | ACM Transactions on Graphics [Электронный ресурс]. – РЕЖИМ ДОСТУПА: https://dl.acm.org/doi/abs/10.1145/3550454.3555523 – 30.05.2023.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1 Алексеев М.А. Алгоритмы поиска соседних частиц при решении задач гидродинамики с помощью метода SPH / М.А. Алексеев // Известия Московского Государственного Индустриального Университета. – 2011. – № 4 (24).

2 Аружан Б. МОДЕЛИ И МЕТОДЫ 3D СИМУЛЯЦИИ ЖИДКОСТЕЙ / Б. Аружан, М. Толқын // Universum: технические науки. – 2023. – № 2-1 (107). – С. 45-48.

3 Бубченко Е.И. Компьютерное Моделирование Жидкости / Е.И. Бубченко // Modern Science. – 2022. – № 6-4.

4 Дикинсон К. Оптимизация игр в Unity 5. Советы и методы оптимизации игровых приложений / К. Дикинсон Google-Books-ID: 5c1SEAAAQBAJ. – Litres, 2022. – 308 с.

5 Евстигнеев Н.М. Высокоскоростные Параллельные Алгоритмы Решения Задач Механики Сплошной Среды Методом Сглаженных Частиц / Н.М. Евстигнеев, Ф.С. Зайцев, О.И. Рябков // Доклады Академии Наук. – 2014. – Т. 459. – № 3.

6 Иванович Н.Е. Моделирование течений слабосжимаемой жидкости методом сглаженных частиц на графических процессорах / Н.Е. Иванович // Известия Самарского научного центра Российской академии наук. – 2016. – Т. 18. – № 2-3. – С. 936-940.

7 Использование программного приложения unity для 3d-моделирования физических объектов [Электронный ресурс]. – URL: https://elibrary.ru/item.asp?id=49395407 (дата обращения: 30.05.2023).

8 Использование средств аппаратной поддержки для повышения производительности систем 3D-пространственной визуализации / С.А. Зори [и др.] // Информатика И Кибернетика. – 2019. – № 1 (15).

9 Литвинов В.В. Сравнение решений уравнения Навье-Стокса методом сеток и методом сглаженных частиц / В.В. Литвинов, О.И. Литвинова. – Ярославский государственный университет им. П.Г. Демидова, 2020. – С. 99-103.

10 Мельникова В.Г. Тестирование различных методов моделирования внутренних течений несжимаемой жидкости / В.Г. Мельникова // Труды Института системного программирования РАН. – 2018. – Т. 30. – № 6. – С. 315-328.

11 Общая характеристика игровых движков [Электронный ресурс]. – URL:https://studbooks.net/2039980/informatika/obschaya_harakteristika_igrovyh_dvizhkov (дата обращения: 30.05.2023).

12 Потапов И.И. Исследование влияния двух геометрических параметров на точность решения гидростатической задачи методом гидродинамики сглаженных частиц / И.И. Потапов, О.В. Решетникова // Компьютерные Исследования И Моделирование. – 2021. – Т. 13. – № 5.

13 Потапов И.И. О влиянии параметров ядра в SPH-методах на точность получаемого решения / И.И. Потапов, О.В. Решетникова. – Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Московский авиационный институт (национальный исследовательский университет)», 2021. – С. 72.

14 Свитецкий, М.А. Разработка многопользовательской мобильной игры / М.А. Свитецкий // «День науки»: материалы XXX научной конференции Амурского государственного университета – Благовещенск: АмГУ, 2021. – С. 36-38.

15 Свитецкий, М.А. Разработка системы симуляции воды для видеоигр //СОВРЕМЕННАЯ НАУКА, ОБЩЕСТВО И ОБРАЗОВАНИЕ. – 2023. – С. 43-47

16 Свитецкий, М.А. Разработка системы симуляции воды для видеоигр // Молодежь XXI века: шаг в будущее: материалы XXIII региональной научно-практической конференции (24 мая 2022 года) – Благовещенск: Изд-во ДальГАУ, 2022. – С. 213-215

17 Сиротина И.К. Методы Оптимизации Графических Файлов Видеоигр / И.К. Сиротина, А.А. Мышковский, К.Э. Остапчук // Актуальные Научные Исследования В Современном Мире. – 2021. – № 2- – 2 (70).

18 Субстанциальная производная и ее использование в гидравлике / О.И. Зайцев [и др.] // Строительство Уникальных Зданий И Сооружений. – 2013. – № 8 (13).

19 Суравикин А.Ю. Моделирование и визуализация несжимаемых жидкостей методом сглаженных частиц на графическом процессоре : кандидат технических наук / А.Ю. Суравикин. – Омск, 2012.

20 Физический движок / Текст : электронный // Википедия / Page Version ID: 121583295. – 2022. – URL: https://ru.wikipedia.org/w/index.php?title=%D0%A4%D0%B8%D0%B7%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B4%D0%B2%D0%B8%D0%B6%D0%BE%D0%BA&oldid=121583295 (дата обращения: 30.05.2023).

21 Храпов С.С. Численное моделирование гидродинамических аварий: размыв дамб и затопление территорий / С.С. Храпов // Вестник Санкт-Петербургского университета. Математика. Механика. Астрономия. – 2023. – Т. 10. – Численное моделирование гидродинамических аварий. – № 2. – С. 357-373.

22 Шарыгин А.Ю. Основные Процессы Оптимизации Мобильной Игры На Unity / А.Ю. Шарыгин, С.П. Кайгордов, С.П. Кайгородов // Аллея Науки. – 2018. – Т. 5. – № 5 (21).

23 Шейдеры и эффекты в Unity. Книга рецептов — Кенни Ламмерс [Электронный ресурс]. – URL: https://www.litres.ru/book/kenni-lammers/sheydery-i-effekty-v-unity-kniga-receptov-22998602/ (дата обращения: 30.05.2023).

24 Эбрахим А.М. Эмуляция Физической Камеры В Unity3d Для Использования В Приложениях Визуальной Одометрии И 3d-Реконструкции / А.М. Эбрахим // Инновационные Научные Исследования. – 2021. – № 3- – 1 (5).

25 A Parallel SPH Implementation on Multi-Core CPUs / M. Ihmsen [et al.] // Computer Graphics Forum. – 2011. – Vol. 30. – № 1. – P. 99-112.

26 A Physically Consistent Implicit Viscosity Solver for SPH Fluids / M. Weiler [et al.] // Computer Graphics Forum. – 2018. – Vol. 37. – № 2. – P. 145-155.

27 Anastasov G. Dynamic surface rendering of water in liquid and solid states in real-time application / G. Anastasov, S. Tomova.

28 Chen F. Meshfree simulation of multiphase flows with SPH family methods / F. Chen.

29 Comparison between the Lagrangian and Eulerian Approach for Simulating Regular and Solitary Waves Propagation, Breaking and Run-Up / D. De Padova [et al.] // Applied Sciences. – 2021. – Vol. 11. – № 20. – P. 9421.

30 Fast Octree Neighborhood Search for SPH Simulations | ACM Transactions on Graphics [Электронный ресурс]. – URL: https://dl.acm.org/doi/abs/10.1145/3550454.3555523 (дата обращения: 30.05.2023).

31 Fluid-flow modeling and stability analysis of communication networks : 20th IFAC World Congress / N. Espitia [et al.] // IFAC-PapersOnLine. – 2017. – Vol. 50. – № 1. – P. 4534-4539.

32 Fraser K. Adaptive smoothed particle hydrodynamics neighbor search algorithm for large plastic deformation computational solid mechanics. 13th International LS-DYNA Users Conference //Dearborn, MI: LSTC. – 2014. Fu L. Numerical methods for computational fluid dynamics - a new ENO paradigm and a new domain decomposition method / L. Fu. – P. 152.

33 Gazebo Fluids: SPH-based simulation of fluid interaction with articulated rigid body dynamics / E. Angelidis [и др.] // 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). – 2022. – Gazebo Fluids. – C. 11238-11245.

34 Giraldo G. CFD Modeling, Analysis & CFD Simulation For Beginners [Электронный ресурс]. – URL: https://www.simscale.com/blog/cfd-analysis-for-beginners/ (дата обращения: 30.05.2023).

35 Journal I. IRJET- A GPU Parallel Implementation of Bitonic Sort using CUDA / I. Journal // IRJET. – 2021.

36 Lind S.J. Review of smoothed particle hydrodynamics: towards converged Lagrangian flow modelling / S.J. Lind, B.D. Rogers, P.K. Stansby // Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences. – 2020. – Т. 476. – Review of smoothed particle hydrodynamics. – № 2241. – С. 20190801.

37 Maya Help | Bifröst Fluids | Autodesk [Электронный ресурс]. – URL: https://help.autodesk.com/view/MAYAUL/2023/ENU/?guid=GUID-F37B36D7-5ABB-4509-B2E6-9F27A3794DA3 (дата обращения: 30.05.2023).

38 Mrope H.A. A Review on Computational Fluid Dynamics Applications in the Design and Optimization of Crossflow Hydro Turbines / H.A. Mrope, Y.A. Chande Jande, T.T. Kivevele // Journal of Renewable Energy. – 2021. – Vol. 2021. – P. e5570848.

39 Price M.J. C# 8.0 and .NET Core 3.0 – Modern Cross-Platform Development: Build applications with C#, .NET Core, Entity Framework Core, ASP.NET Core, and ML.NET using Visual Studio Code, 4th Edition. C# 8.0 and .NET Core 3.0 – Modern Cross-Platform Development / M.J. Price Google-Books-ID: Qzm8DwAAQBAJ. – Packt Publishing Ltd, 2019. – 819 p.

40 Real-time Sponge and Fluid Simulation / Burkus, Viktória [et al.]. – 2022. – P. 4 pages.

41 Saftly W. Hierarchical octree and k-d tree grids for 3D radiative transfer simulations / W. Saftly, M. Baes, P. Camps // Astronomy & Astrophysics. – 2014. – Vol. 561. – P. A77.

42 SPH Fluids in Computer Graphics / M. Ihmsen [et al.] // Eurographics 2014 - State of the Art Reports. – 2014. – P. 22 pages.

43 SPH Techniques for the Physics Based Simulation of Fluids and Solids / D. Koschier [et al.]. – 2019. – P. 41.

44 Sridhar P. Discretization approaches to model orthogonal cutting with Lagrangian, Arbitrary Lagrangian Eulerian, Particle Finite Element method and Smooth Particle Hydrodynamics formulations : 53rd CIRP Conference on Manufacturing Systems 2020 / P. Sridhar, J.M. Rodríguez Prieto, K.M. de Payrebrune // Procedia CIRP. – 2020. – Vol. 93. – P. 1496-1501.

45 Surface Tension Model Based on Implicit Incompressible Smoothed Particle Hydrodynamics for Fluid Simulation / X. Wang [и др.] // Journal of Computer Science and Technology. – 2017. – Т. 32. – С. 1186-1197.

46 Technologies U. Unity - Manual: Compressing mesh data [Электронный ресурс]. – URL: https://docs.unity3d.com/Manual/mesh-compression.html (дата обращения: 05.06.2023).

47 Technologies U. Unity - Manual: Physics [Электронный ресурс]. – URL:

https://docs.unity3d.com/Manual/PhysicsSection.html (дата обращения: 30.05.2023).

48 Technologies U. Unity - Manual: Shaders [Электронный ресурс]. – URL: https://docs.unity3d.com/Manual/Shaders.html (дата обращения: 30.05.2023).

49 Technologies U. Батчинг вызовов отрисовки (Draw Call Batching) - Unity Manual [Электронный ресурс]. – URL: https://docs.unity3d.com/ru/2019.3/Manual/DrawCallBatching.html (дата обращения: 21.05.2023).

50 Turbulent Details Simulation for SPH Fluids via Vorticity Refinement / S. Liu [et al.] // Computer Graphics Forum. – 2021. – Vol. 40. – № 1. – P. 54-67.

51 Violeau D. Fluid mechanics and the SPH method: theory and applications. Fluid mechanics and the SPH method / D. Violeau. – 1st ed. – Oxford ; New York: Oxford University Press, 2012. – 594 p.

52 Winchenbach R. Multi-Level Memory Structures for Simulating and Rendering Smoothed Particle Hydrodynamics / R. Winchenbach, A. Kolb // Computer Graphics Forum. – 2020. – Vol. 39. – № 6. – P. 527-541.

53 Zhang X. Parallel SPH fluid control with dynamic details / X. Zhang, S. Liu // Computer Animation and Virtual Worlds. – 2018. – Vol. 29. – № 2. – P. e1801.