Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМ. Н. Э. БАУМАНА (национальный исследовательский университет)

УДК _____
№ госрегистрации _____
Инв. №_____

УТВЕРЖДАЮ

_____
головной исполнитель НИР

_____ _____

«_____» _____ 2019 г.

ОТЧЁТ
О НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

по теме:

Программа моделирования движения воды с использованием вокселов
(промежуточный)

Руководитель темы _____ А. С. Кострицкий

Москва 2019

РЕФЕРАТ

Отчет содержит 33 стр., 1 рис., 22 источн., 1 прил.

Это пример каркаса расчётно-пояснительной записки, желательный к использованию в РПЗ проекта по курсу РСОИ .

СОДЕРЖАНИЕ

Введение 5

1 Аналитический раздел 6

1.1 Физическая модель 6

1.2 Существующие подходы к симуляции жидкостей 6

1.3 Анализ методов визуализации жидкостей 8

1.3.1 Объёмный алгоритм маршрующих лучей 8

1.3.2 Реконструкция поверхности 9

1.3.3 Оптимизации воксельного способа решения задачи объёмного рендеринга 10

1.3.4 Вывод 10

1.4 Анализ алгоритмов визуализации вокселов 11

1.4.1 Анализ алгоритмов удаления невидимых линий и поверхностей 11

1.4.1.1 Алгоритм с использованием Z-буферизации 11

1.4.1.2 Алгоритм с использованием маршрующих лучей и функции знака расстояния 12

1.4.1.3 Вывод 13

1.4.2 Анализ алгоритмов закрашивания 13

1.4.2.1 Закраска по Ламберту 14

1.4.2.2 Закраска по Гуро 14

1.4.2.3 Закраска по Фонгу 14

1.4.2.4 Вывод 14

1.5 Вывод 15

2 Конструкторский раздел 16

2.1 Архитектура приложения 16

2.1.1 Модуль наблюдателя 17

2.1.2 Модуль построения виртуального мира 17
2.1.3 Модуль пользовательского интерфейса 18
2.1.4 Модуль рендеринга сцены 18
2.1.4.1 Оптимизированный алгоритм марширующих кубов 18
2.1.4.2 Алгоритм с использованием Z-буферизации 19
2.1.4.3 Закрашивание по Ламберту 19
2.2 Вывод 20
3 Технологический раздел 21
3.1 Требования к программному обеспечению 21
3.2 Используемые технологии 21
3.3 Листинги кода 22
3.3.1 Клеточный автомат 22
3.3.2 Реконструкция поверхности 24
3.3.3 Рендеринг вокселов 24
3.4 Вывод 26
4 Экспериментально-исследовательский раздел 27
4.1 Исследование характеристик программы 27
4.2 Примеры использования программы 27
4.3 Выводы 28
Заключение 29
Список использованных источников 30
Приложение А Рисунки 33

ВВЕДЕНИЕ

Визуализация различных явлений становится всё более важной во множестве инженерных областей знаний. Задача объёмного рендеринга имеет большое значение, например, в визуализации данных компьютерной и магнитно-резонансной томографии[12]. Интерактивная 3D симуляция позволяет учёным ясно воспринимать и оценивать результаты собственных исследований.

В настоящее время для этого используются различные вычислительные техники обработки и графического представления экспериментальных данных[6].

В данной работе рассматривается 3D симуляция воды. Симуляция жидкостей, в целом, является примером того, что получаемые о них сведения без соответствующего 3D изображения довольно сложны для человеческого восприятия.

Целью работы является разработка программного продукта для моделирования движения воды с использованием воксельной графики. Для достижения поставленной цели необходимо решить следующие задачи:

— проанализировать существующие методы моделирования движения жидкостей и методы рендеринга с помощью вокселов;

— спроектировать программное обеспечение, симулирующее поведение воды;

— реализовать программу и проверить её работоспособность.

1 Аналитический раздел

В данном разделе производится анализ методов вычислений характеристик жидкостей и их преобразований в графический формат.

Далее рассматриваются идеи применения данных методов к симуляции жидкостей и существующие решения в этой области.

1.1 Физическая модель

Жидкости моделируются как векторное поле скорости жидкости и скалярное поле плотности. Движение задаётся уравнениями Навье-Стокса [1].

Далее рассматривается только движение воды (несжимаемой жидкости) в условиях постоянной температуры.

Тогда уравнения Навье-Стокса в векторной форме принимают следующий вид:

$$\frac{\partial \vec{v}}{\partial t} + \vec{v} \cdot \nabla \vec{v} = \vec{F} - \frac{1}{\rho} \nabla p + \eta \Delta \vec{v}. \tag{1.1}$$

В уравнении 1.1 $\vec{v}$ - скорость частицы воды, t - время, $\vec{F}$ - внешняя удельная сила, p - давление, $\eta = \frac{\mu}{\rho}$ - кинематический коэффициент вязкости, ∇ - оператор Гамильтона, Δ - оператор Лапласа.

Данная физическая модель лежит в основе многих подходов симуляции жидкостей[12]. Их обзор приведён далее.

В статистической физике модель поведения частиц жидкости описывается кинетическим уравнением Больцмана. Данная модель применима для систем, где есть ограничения на малую скорость частиц[13].

1.2 Существующие подходы к симуляции жидкостей

В вычислениях поведения жидкости необходимо представить физическую модель в дискретном виде. Данную проблему решает вычислительная гидродинамика - совокупность теоретических, экспериментальных

и численных методов, предназначенных для моделирования потоковых процессов.

Наиболее распространёнными методами описания характеристик жидкости в вычислительной гидродинамике являются:

— сеточные методы Эйлера;

— метод гидродинамики сглаженных частиц;

— методы, основанные на турбулентности;

— метод решёточных уравнений Больцмана.[12]

Сеточные методы Эйлера являются наиболее простым решением симуляции жидкостей Они заключаются в поиске решения задачи Коши для функций, заданных таблично. Для каждого узла функции уровня жидкости требуется вычисление значений разложений Тейлора в их окрестностям. Решение задачи Коши в данном случае аппроксимирует решение уравнений Навье-Стокса[15].

На основе сеточных методов Эйлера основан способ, который аппроксимирует решение уравнений Навье-Стокса при помощи клеточного автомата[4]. Жидкость представляется в виде трёхмерной сетки. Для каждой "клетки"жидкости в каждом новом поколении вычисляется новое состояние - кинетическая энергия и уровень жидкости - на основе состояний "соседей фон Неймана"этой ячейки жидкости.

Метод гидродинамики сглаженных частиц и методы, основанные на турбулентности, заключаются в выборе размера частицы ("длины сглаживания"), на котором их свойства "сглаживаются"посредством функции ядра или интерполяции, и решения уравнений Навье-Стокса с учётом вязкости и плотности. Это позволяет эффективно моделировать поведение жидкостей, газов и даже использовать в астрофизике[16].

Метод решёточных уравнений Больцмана основан на кинетическом уравнении Больцмана, упомянутом ранее. Этот метод поддерживает многофазные жидкости, наличие теплопроводности и граничные условия на макроскопическом уровне[14].

В данной работе для симуляции воды используется метод клеточного автомата, поскольку является наименее вычислительно сложной аппроксимацией уравнений Навье-Стокса[4].

1.3 Анализ методов визуализации жидкостей

Основными требованиями к методам симуляции жидкостей со стороны компьютерной графики являются визуальная правдоподобность и скорость анимации.

Исходя из этих условий, в компьютерной графике используются специально модифицированные и оптимизированные методы, основанные на указанных ранее. При этом решается задача объёмного рендеринга - на каждый кадр требуется найти проекцию трёхмерного дискретного набора данных о жидкости[17].

В настоящее время существует всего 2 принципиально различных техники объёмного рендеринга: объёмный алгоритм марширующих лучей и реконструкция поверхности[12].

1.3.1 Объёмный алгоритм марширующих лучей

Данная техника состоит в том, чтобы создать виртуальный экран и далее проследить путь луча до тех пор, пока он проходит сквозь анимируемые объекты. В итоге луч позволяет определить цвет каждого отдельного пикселя. Настоящий алгоритм трассировки лучей имеет значительную вычислительную сложность, поэтому его часто заменяют на правдоподобные аппроксимации[12].

Одним из способов аппроксимации является трассировки лучей путём замены полигонов на вокселы. Это позволяет существенно повышать производительность анимации за счёт возможностей параллелизма и снижения вычислительной сложности за счёт упрощения функции знака расстояния между источником света и гранями воксела[12]. Недостатком данного подхода является то, что для хранения вокселов требуется большее количество памяти, чем для хранения полигонов.

Другим популярным методом является метод срезов объёмных текстур так же, как и воксельный, является математическим упрощением алгоритма трассировки лучей[9]. Данные сцены представляются в виде 3D текстуры. Рендеринг производится с самого дальнего по отношению к наблюдателю слоя текстуры. В итоге видимыми являются ближайшие из частей слоя. В качестве составляющих срезов могут быть использованы как вокселы, так и полигоны. Этот способ оптимизирован аппаратно для некоторых графических процессоров и в этом случае имеет преимущество перед воксельным методом по времени работы.

1.3.2 Реконструкция поверхности

Реконструкцией поверхности, или трёхмерной реконструкцией, называют процесс получения облика реальных объектов.

В качестве входных данных получают множество точек, характеризующее объект. Далее выбирают точки на поверхности объекта так, что получают полигоны, которые имеют необходимую конфигурацию для аппроксимации формы поверхности в её видимой части. Набор результирующих полигонов - реконструированная модель.

Основным алгоритмом, реализующим этот метод, является алгоритм марширующих кубов. Он широко используется в компьютерной и магнитно-резонансной томографии[18].

Идея алгоритма состоит в том, что для кубов, находящихся на поверхности объекта, известно, какие точки лежат внутри объекта и какие снаружи. Это позволяет выбрать приближающий полигон так, чтобы его вершины находились на отрезках, соединяющих вершины куба, в примерных точках реального пересечения с объектом. Также можно объединять полигоны на одной поверхности для оптимизации их хранения.

Таким образом, однажды применив алгоритм, можно использовать объект в анимации, если он не изменяет форму поверхности.

Для предметов, теряющих форму (например, жидкостей), требуется повторное вычисление реконструированной поверхности. Подобные вычисления имеют значительную сложность, поэтому на практике, как правило, не используются[12]. Как и в случае с методом марширующих лучей, существуют аппаратные ускорители для данного решения[11].

1.3.3 Оптимизации воксельного способа решения задачи объёмного рендеринга

Существуют случаи, в которых система объёмного рендеринга получает на вход трёхмерные данные, в которых есть области, не требующие отрисовки. В подобных ситуациях следует пропускать вычисление результирующего изображения для пустого пространства[2].

Для хранения вокселов можно использовать различные структуры данных. Наиболее простой является трёхмерный массив вокселов. При хранении вокселов в виде массива может требоваться значительное количество памяти, даже если исходная информация достаточно однородная. Поэтому для оптимизации хранения применяют такие структуры данных, как октодерево и разреженное бинарное дерево[2].

Использование иерахических структур данных имеет следующий недостаток - они статичны, и их использование для моделей, теряющих форму, требует перестроение всего дерева[2].

Поскольку вода теряет свою форму, были предложены оптимизации не для хранения вокселов, а для обработки лучей и самих данных симуляции. Данные оптимизации основаны на принципах обработки чисел с плавающей запятой на аппаратном обеспечении, а также на возможности ускорения вычислений за счёт параллелизма[12].

1.3.4 Вывод

В данной работе используется метод реконструкции поверхности, который используется совместно с подходом марширующих лучей. В ка-

честве способа реконструкции используется оптимизация марширующих кубов для поиска больших прямоугольников на воксельных поверхностях.

В качестве структуры данных для хранения вокселов предлагается использовать хеш-таблицу, в которой хешами являются координаты воксела. Если воксел присутствует в сцене, то для него найдётся соответствующее логическое значение из хеш-таблицы. Данный способ позволяет избежать хранения в памяти пустого пространства сцены.

1.4 Анализ алгоритмов визуализации вокселов

Ранее рассматриваются методы для описания сцены из вокселов, которая представляет собой результат симуляции воды. Для их визуализации выбран метод реконструкции поверхности совместно с подходом алгоритма марширующих лучей. В данном методе предусмотрена возможность использования множества стандартных алгоритмов компьютерной графики для удаления невидимых линий и поверхностей, а также закрашивания [12]. Далее производится анализ этих алгоритмов визуализации.

1.4.1 Анализ алгоритмов удаления невидимых линий и поверхностей

В данном подразделе рассматриваются алгоритмы для удаления невидимых линий и поверхностей, пригодные для использования в ранее указанном методе:

— алгоритм с использованием Z-буферизации;

— алгоритм с использованием марширующего луча и функции знака расстояния.

1.4.1.1 Алгоритм с использованием Z-буферизации

Далее описывается суть алгоритма с использованием Z-буферизации.

а) Создаётся двумерный массив - Z-буфер, каждый элемент которого соответствует пикселю изображения.

б) Полученную с помощью реконструкции поверхности триангулированную сетку разбивают на треугольники и находят их проекции на картинную плоскость.

в) Для каждого спроецированного треугольника вычисляется объемлющий прямоугольник.

г) Для каждого пикселя внутри объемлющего прямоугольника вычисляется значение глубины на основе координат вершин спроецированного треугольника.

д) Если значение Z-буфера для данного пикселя меньше, чем значение глубины в нём, то в Z-буфер записывается данное значение глубины. В буфер изображения записывается данный пиксель.

На выходе алгоритма - буфер изображения, каждый пиксель которого окрашен в цвет ближайшего треугольника.

Достоинства алгоритма:

— возможна параллельная обработка треугольников;

— не требуется сортировка вокселов;

— возможно аппаратное ускорение для чтения и записи из Z-буфера.

Недостатком является значительное использование памяти, но для современных вычислительных систем это часто оказывается приемлемым[19].

1.4.1.2 Алгоритм с использованием марширующих лучей и функции знака расстояния

Для данного алгоритма требуется задание функции для каждого объекта сцены, значение которой положительно, если выбранная точка находится снаружи объекта сцены, и отрицательно, если выбранная точка находится внутри объекта сцены.

Далее для каждого луча последовательно выбирают точки до тех пор, пока значение функции знака положительно. Если её значение стало отрицательным, то найдено пересечение, следовательно, данную точку можно записывать в результат.

Достоинства алгоритма:

— возможно определение реалистичного цвета поверхности на основе пройденного расстояния луча;

— возможно использование многих источников света;

— возможно использование обратного алгоритма марширующих лучей(от объектов к картинной плоскости), в котором допустима параллельная обработка лучей.

Недостатки алгоритма:

— зависит от выбора функции знака расстояния;

— вычислительно сложнее, чем алгоритм с использованием Z-буферизации[9].

1.4.1.3 Вывод

В результате выбран алгоритм с использованием Z-буферизации, поскольку временные затраты на вычисление расстояния в алгоритме марширующих лучей больше.

1.4.2 Анализ алгоритмов закрашивания

Для создания более реалистичного изображения следует использовать некоторую модель освещения. Наиболее точным методом является трассировка лучей[9], но, как было описано ранее, данная модель требует значительных вычислительных ресурсов.

Далее будут рассмотрены некоторые из возможных аппроксимаций для закрашивания поверхностей вокселов.

1.4.2.1 Закраска по Ламберту

Суть метода постоянного закрашивания (по Ламберту) заключается в том, что на каждом полигоне определяется освещённость в произвольной точке, и полученное значение используется для всего полигона[10]. В результате изображение имеет явный полигональный характер. С точки зрения закраски полигонов, которые находятся на поверхности воксела, - это самый эффективный метод закрашивания с учётом освещения[10].

1.4.2.2 Закраска по Гуро

В отличие от закраски по Ламберту, в закраске по Гуро используется билинейная интерполяция интенсивностей света в каждой вершине полигона. Данный метод устраняет дискретность интенсивностей и работает верно в диффузной модели освещения. Зеркальное освещение может быть нарушено, так как происходит усреднение векторов нормалей[10].

1.4.2.3 Закраска по Фонгу

В данном методе производится билинейная интерполяция векторов нормалей в каждой из вершин, что позволяет достичь более реалистичного изображения, чем в методе Гуро. Из недостатков выделяют большую трудоёмкость, чем в методе Гуро, а также существуют случаи, в которых модель Фонга создаёт эффект полос Маха[10].

1.4.2.4 Вывод

В дальнейшем используется метод закраски по Ламберту, так как он наиболее эффективен по времени при закраске граней вокселов среди всех методов закраски с учётом освещения.

1.5 Вывод

Исходя из полученных сведений о достоинствах и недостатках каждого из методов объёмного рендеринга в дальнейшем рассматривается реализация с помощью оптимизированного метода маршируюших кубов. Она обеспечивает возможность изменения формы жидкости и эффективное удаление невидимых линий и поверхностей посредством Z-буферизации.

2 Конструкторский раздел

В данном разделе описано проектирование метода рендеринга воды с помощью вокселов с указанием соответствующих схем алгоритмов.

Разработанное приложение должно решать задачу синтеза сложного динамического изображения.

Эту задачу принято разделять на следующие этапы:

— разработка трёхмерной математической модели синтезируемой визуальной обстановки;

— задание положения наблюдателя, картинной плоскости, размеров окна вывода, значений управляющих сигналов;

— определение операторов, осуществляющих пространственное перемещение объектов визуализации;

— преобразования координат объектов в координаты наблюдателя;

— отсечение объектов сцены по границам пирамиды видимости;

— вычисление двумерных перспективных проекций объектов на картинную плоскость;

— удаление невидимых линий и поверхностей при заданном положении наблюдателя;

— закрашивание и затенение видимых объектов сцены;

— вывод полученного полутонового изображения на экран растрового дисплея.

2.1 Архитектура приложения

В соответствии с каждым этапом синтеза изображения были выделены следующие модули программы:

— наблюдатель (камера) и его управляющие сигналы;

— модуль построения виртуального мира, включающий математической модели воды и земли;

— модуль пользовательского интерфейса;

— модуль рендеринга сцены.

2.1.1 Модуль наблюдателя

Структура данных наблюдателя содержит в себе вектор позиции в трёхмерном пространстве и углы крена, тангажа и рысканья.

Для наблюдения сцены строится одноточечная перспективная проекция с помощью следующей матрицы: $\begin{pmatrix} \frac{near \times 2}{right-left} & 0 & \frac{right+left}{right-left} & 0 \\ 0 & \frac{near \times 2}{top-bottom} & \frac{top+bottom}{top-bottom} & 0 \\ 0 & 0 & -\frac{far+near}{far-near} & -\frac{far \times 2 \times near}{far-near} \\ 0 & 0 & -1 & 0 \end{pmatrix}$,где $fovy = 0.785$, $aspect - ratio = \frac{4}{3}$, $near = 1$, $far = 80$, $top = near \times tg(\frac{fovy}{2})$, $left = -right$, $right = top \times aspect - ratio$, $bottom = -top$.

2.1.2 Модуль построения виртуального мира

Для построения виртуальной поверхности земли предлагается использовать трёхмерный шум Перлина. Поскольку земля может не менять свою форму во время симуляции, то следует генерировать данную поверхность однократно.

Для воды необходимо определить клеточный автомат. Он должен учитывать скорость частиц воды, давление и сопротивление среды($friction$). При этом должны учитываться коллизии с землёй($terrain$). Для этого выделим два двумерных массива размерности мира: в одном содержится текущий уровень воды в данной точке ($water$), а в другом - кинетическая энергия воды в данной точке ($energy$). Тогда для каждой ячейки (i, j) автомата во время перехода в очередное состояние выполняются следующие действия:

а) определить давление с четырёх сторон ($left$, $right$, $front$, $back$)ячейки (если ячейки нет, то давление равно нулю):

1) $left = terrain_{i-1,j} + water_{i-1,j} + energy_{i-1,j}$;

2) $right = terrain_{i+1,j} + water_{i+1,j} - energy_{i+1,j}$;

3) $front = terrain_{i,j+1} + water_{i,j+1} - energy_{i,j+1}$;

4) $back = terrain_{i,j-1} + water_{i,j-1} + energy_{i,j-1}$;

б) определить величину изменения уровня воды в данной ячейке на основе заданных давлений, сопротивления среды и уровня земли;

в) обновить состояние клеточного автомата.

2.1.3 Модуль пользовательского интерфейса

В пользовательском интерфейсе необходимо реализовать:

— холст для отображения картинной плоскости;

— элементы интерфейса для управления наблюдателем.

2.1.4 Модуль рендеринга сцены

В данном подразделе описаны основные алгоритмы, используемые для объёмного рендеринга воксельной сцены.

2.1.4.1 Оптимизированный алгоритм марширующих кубов

На вход алгоритму подают последовательность вокселов, на выходе - последовательность треугольников на поверхности заданной структуры.

Алгоритм выглядит следующим образом:

а) заполнить хеш-таблицу признаком присутствия вокселов;

б) для каждого присутствующего воксела в сцене определить внешние поверхности;

в) если внешние поверхности соседних вокселов лежат в одной плоскости - объединить их;

г) полученный набор поверхностей триангулировать, на основе данных о границах и нормалях к поверхностям.

2.1.4.2 Алгоритм с использованием Z-буферизации

Задаётся массив размерности картинной плоскости для хранения глубины пикселя. Изначально он инициализирован минимальным значением типа данных.

В процессе рендеринга треугольников для каждого треугольника определяются прямоугольные объемлющие оболочки.

По всем пикселям внутри оболочки выполяются следующие действия:

а) определить глубину пикселя в данной точке из барицентрических координат;

б) если глубина пикселя меньше текущего значения в массиве глубин, то в буфер изображения записываем вычисленный цвет треугольника, а в буфер глубины - новое значение глубины.

2.1.4.3 Закрашивание по Ламберту

На вход алгоритм получает позицию вершины, цвет её материала и нормаль. На выходе получаем цвет RGBA.

Расчёт интенсивности в вершине производится по формуле: $I = I_{src} * cos(\alpha)$, где I_{src} - интенсивность источника, α - угол между нормалью и вектором направления света.

Интенсивность источника состоит из двух компонент - интенсивности зеркального отражения и интенсивности диффузного.

К интенсивности добавляется интенсивность внешнего постоянного света.

Посредством умножения составляющих частей (красной, зелёной и синей) света на соответствующие части цвета материала получаем результирующий цвет.

2.2 Вывод

В данной части были определены основные модули программы, которые необходимо реализовать, и их состав.

3 Технологический раздел

В данном разделе описаны требуемые средства и подходы к реализации ПО по ранее указанным методам.

3.1 Требования к программному обеспечению

Разработанное ПО должно моделировать движение воды с использованием вокселов.

Моделирование движения должно осуществляться с использованием операций переноса, масштабирования и поворота.

3.2 Используемые технологии

Для реализации ПО выбран язык Clojure. Данный язык программирования является компилируемым и, одновременно с этим, динамическим. Clojure - преимущественно язык функционального программирования, который поддерживает нативный доступ к Java фреймворкам [20].

Данные свойства языка позволяют вести разработку приложений с помощью REPL (Read-Eval-Print Loop), модифицируя их во время выполнения. Это может быть полезно для добавления новых функциональностей в программу без её повторной сборки [21].

Для создания пользовательского интерфейса используется Java библиотека Swing. Swing предоставляет широкий набор компонентов для создания графического пользовательского интерфейса. Данные компоненты полностью реализованы на Java, поэтому их внешний вид не зависит от платформы.

Для организации проекта на Clojure и для автоматизации сборки используется Leiningen. Все конфигурации для сборки приложения, запуска REPL и описания зависимостей управляются с помощью этого модуля[22]. Результатом сборки является JAR файл, который можно запустить на любом компьютере, где установлено соответствующее JRE.

Для оптимизации работы алгоритмов рендеринга использованы подсказки типа - механизм языка Clojure, позволяющий избежать рефлексивного обращения к JVM. Также используются средства параллельной обработки ленивых последовательностей Clojure - "pmap".

Для оптимизации хранения некоторых воксельных структур использована мемоизация - механизм Clojure, который позволяет запоминать результаты выполнения функций.

3.3 Листинги кода

В данном разделе будут представлены листинги реализации наиболее существенных модулей программы.

3.3.1 Клеточный автомат

В данном подразделе указан листинг кода, относящийся к реализации клеточного автомата симуляции воды.

В листинге 3.3.1 представлена реализация обновления состояния клеточного автомата.

```
(let [left-pressure (if (> i 0)
                      (+ (aget terrain-row-prev j)
                         (aget water-row-prev j)
                         (aget energy-row-prev j))
                      0.0)
      right-pressure (if (< i (dec dim))
                       (+ (aget terrain-row-next j)
                          (aget water-row-next j)
                          (- (aget energy-row-next j)))
                       0.0)
      back-pressure (if (> j 0)
                      (+ (aget terrain-row-cur (dec j))
                         (aget water-row-cur (dec j))
                         (aget energy-row-cur (dec j)))
                      0.0)
      front-pressure (if (< j (dec dim))
                       (+ (aget terrain-row-cur (inc j))
                          (aget water-row-cur (inc j))
                          (- (aget energy-row-cur (inc j))))
                       0.0)]
  (if (and (> i 0)
           (> (+ (aget terrain-row-cur j)
                 (aget water-row-cur j)
```

```
                    (aget energy-row-cur j))
                 left-pressure))
          (let [flow (/ (min (aget water-row-cur j)
                             (+ (aget terrain-row-cur j)
                                (aget water-row-cur j)
                                (- (aget energy-row-cur j))
                                (- left-pressure)))
                        flow-div)]
            (aset water-row-prev j (+ (aget water-row-prev j) flow))
            (aset water-row-cur j (+ (aget water-row-cur j) (- flow)))
            (aset energy-row-prev j (* (aget energy-row-prev j) (- 1.0 friction)))
            (aset energy-row-prev j (+ (aget energy-row-prev j) (- flow)))))
        (if (and (< i (dec dim))
                 (> (+ (aget terrain-row-cur j)
                       (aget water-row-cur j)
                       (aget energy-row-cur j))
                    right-pressure))
          (let [flow (/ (min (aget water-row-cur j)
                             (+ (aget terrain-row-cur j)
                                (aget water-row-cur j)
                                (aget energy-row-cur j)
                                (- right-pressure)))
                        flow-div)]
            (aset water-row-next j (+ (aget water-row-next j) flow))
            (aset water-row-cur j (+ (aget water-row-cur j) (- flow)))
            (aset energy-row-next j (* (aget energy-row-next j) (- 1.0 friction)))
            (aset energy-row-next j (+ (aget energy-row-next j) flow))))
        (if (and (> j 0)
                 (> (+ (aget terrain-row-cur j)
                       (aget water-row-cur j)
                       (- (aget energy-row-cur j)))
                    back-pressure))
          (let [flow (/ (min (aget water-row-cur j)
                             (+ (aget terrain-row-cur j)
                                (aget water-row-cur j)
                                (- (aget energy-row-cur j))
                                (- back-pressure)))
                        flow-div)]
            (aset water-row-cur (dec j) (+ (aget water-row-cur (dec j)) flow))
            (aset water-row-cur j (+ (aget water-row-cur j) (- flow)))
            (aset energy-row-cur (dec j) (* (aget energy-row-cur (dec j)) (- 1.0 friction)))
            (aset energy-row-cur (dec j) (+ (aget energy-row-cur (dec j)) (- flow)))))
        (if (and (< j (dec dim))
                 (> (+ (aget terrain-row-cur j)
                       (aget water-row-cur j)
                       (aget energy-row-cur j))
                    front-pressure))
          (let [flow (/ (min (aget water-row-cur j)
                             (+ (aget terrain-row-cur j)
                                (aget water-row-cur j)
                                (aget energy-row-cur j)
                                (- front-pressure)))
                        flow-div)]
            (aset water-row-cur (inc j) (+ (aget water-row-cur (inc j)) flow))
            (aset water-row-cur j (+ (aget water-row-cur j) (- flow)))
            (aset energy-row-cur (inc j) (* (aget energy-row-cur (inc j)) (- 1.0 friction)))
```

```
80              (aset energy-row-cur (inc j) (+ (aget energy-row-cur (inc j)) flow)))))
```

3.3.2 Реконструкция поверхности

В данном подразделе указан листинг, относящийся к реконструкции поверхности сцены.

В листинге 3.3.2 представлена реализация реконструкции поверхности.

```
1  (defn generate-voxel-mesh
2    "Generate mesh that surrounds our voxel model"
3    [voxels]
4    (let
5      [lookup-table (fill-lookup-table voxels)
6       exposed-faces (find-exposed-faces voxels lookup-table)
7       plane-faces (.entrySet exposed-faces)
8       faces (pmap combine-faces-single-plane plane-faces)
9       triangles (flatten (pmap triangulate-faces-single-plane faces))]
10
11     (Mesh. triangles)))
```

3.3.3 Рендеринг вокселов

В данном подразделе указаны листинги, относящиеся к растеризации сцены.

В листинге 3.3.3 описана реализация растеризации воксельной сцены.

```
1  (defn draw-triangles
2  [canvas triangles mvp]
3  (let
4  [viewport [(.getWidth canvas) (.getHeight canvas)]
5   width ^long (viewport 0)
6   height ^long (viewport 1)
7   neg-inf Double/NEGATIVE_INFINITY
8   len ^long (* width height)
9   z-buffer ^doubles (hiphip/amake Double/TYPE [_ len] neg-inf)]
10
11 (doall (pmap (fn [^Triangle triangle]
12                (let [v1 ^Vertex (:v1 triangle)
13                      v2 ^Vertex (:v2 triangle)
14                      v3 ^Vertex (:v3 triangle)
15                      p1 (camera/project-to-screen (:position v1) mvp viewport)
16                      p2 (camera/project-to-screen (:position v2) mvp viewport)
17                      p3 (camera/project-to-screen (:position v3) mvp viewport)
18                      p1-x ^double (m/mget p1 0)
```

```
                  p1-y ^double (m/mget p1 1)
                  p1-z ^double (m/mget p1 2)
                  p2-x ^double (m/mget p2 0)
                  p2-y ^double (m/mget p2 1)
                  p2-z ^double (m/mget p2 2)
                  p3-x ^double (m/mget p3 0)
                  p3-y ^double (m/mget p3 1)
                  p3-z ^double (m/mget p3 2)
                  c ^int (colorize v2)
                  min-x ^long (long (max 0.0 (Math/ceil (min p1-x p2-x p3-x))))
                  max-x ^long (long (min (dec width) (Math/floor (max p1-x p2-x p3-x))))
                  min-y ^long (long (max 0.0 (Math/ceil (min p1-y p2-y p3-y))))
                  max-y ^long (long (min (dec height) (Math/floor (max p1-y p2-y p3-y))))
                  area ^double (+ (* (- p1-y p3-y) (- p2-x p3-x)) (* (- p2-y p3-y) (- p3-x
                      p1-x)))]

              (doall (p/pmap (fn [^long y]
                               (doall
                                 (map (fn [^long x]
                                        (let [b1 ^double (/ (+ (* (- y p3-y) (- p2-x p3-x))
                                                               (* (- p2-y p3-y) (- p3-x x)))
                                                            area)
                                              b2 ^double (/ (+ (* (- y p1-y) (- p3-x p1-x))
                                                               (* (- p3-y p1-y) (- p1-x x)))
                                                            area)
                                              b3 ^double (/ (+ (* (- y p2-y) (- p1-x p2-x))
                                                               (* (- p1-y p2-y) (- p2-x x)))
                                                            area)
                                              depth ^double (+ (* b1 p1-z) (* b2 p2-z) (* b3
                                                  p3-z))
                                              z-index ^long (+ (* y width) x)]
                                          (if (< (aget z-buffer z-index) (+ depth 10e-5))
                                            (do
                                              (.setRGB canvas x y c)
                                              (aset z-buffer z-index depth)))))
                                      (range min-x (inc max-x)))))
                             (range min-y (inc max-y))))))
          triangles))))

(defn draw-mesh [canvas ^Mesh mesh mvp]
(let [triangles (:triangles mesh)]+ 0.1
(draw-triangles canvas triangles mvp)))

(defn draw-voxels
[^BufferedImage canvas voxels ^Camera camera]
(let [mesh (voxel/generate-voxel-mesh voxels)
   mvp (camera/model-view-projection-matrix (camera/perspective (:position camera))
                                            (camera/get-view-matrix camera)
                                            model-matrix)]
(draw-mesh canvas mesh mvp)))
```

В листинге 3.3.3 описана реализация получения цвета по Ламберту.

```
(defn colorize [v1]
  (let [v1-xyz (:position v1)
        c (:color v1)
```

```
4         r (.getRed c)
5         g (.getGreen c)
6         b (.getBlue c)
7         v1-normal (vec/normalize (voxel/voxel-normal (:normal v1)))
8         light-direction-v1 (vec/normalize (vec/sub @lights/light v1-xyz))
9         view-direction-v1 (vec/normalize (vec/sub v1-xyz))
10        reflect-v1 (vec/reflect (vec/sub light-direction-v1) v1-normal)
11        diffuse-v1 (vec/scale lights/diffuse-albedo (max (vec/dot v1-normal light-direction-v1)
              0.0))
12        specular-v1 (vec/scale lights/specular-albedo (Math/pow (max (vec/dot reflect-v1
              view-direction-v1) 0.0) lights/specular-power))
13        modification (vec/add lights/ambient diffuse-v1 specular-v1)]
14
15    (+ (bit-shift-left (int (* 255 (modification 3))) 24)
16       (bit-shift-left (int (* r (modification 0))) 16)
17       (bit-shift-left (int (* g (modification 1))) 8)
18       (int (* b (modification 2))))))
```

3.4 Вывод

В результате было реализовано программное обеспечение, моделирующее поведение воды.

Для написания ПО использован язык программирования Clojure, проведены оптимизации кода с учётом подсказок типа.

4 Экспериментально-исследовательский раздел

В данном разделе проводится апробация и анализ разработанной программы.

Анализ производится с помощью модуля Clojure - time, а также при помощи VisualVM - инструмент профайлинга приложений под JVM.

Использовалось следующее аппаратное обеспечение:

— процессор AMD Ryzen 5 3550h;

— ОЗУ DDR4 16GB.

Приложение запускалось под управлением операционной системы Ubuntu 19.10.

4.1 Исследование характеристик программы

Рассмотрим рендеринг сцены, состоящей из 30170 вокселов земли и 3000 вокселов воды в разрешении 1920×1080.

Среднее время рендеринга одного кадра по результатам 140 измерений составило 3867 мс. При этом по результатам профайлинга утилитой VisualVM получено, что 65.3 процентов данного времени производились математические операции, причём утилизировано 627 потоков.

Таким образом, для ускорения рендеринга требуется использовать более мощное аппаратное обеспечение, а математические операции передавать на GPU.

4.2 Примеры использования программы

В данном разделе представлены примеры использования разработанной программы.

Управление наблюдателем осуществляется либо с помощью интерфейса, либо с помощью клавиш W, A, S, D и стрелок.

На А.1 представлен снимок основного экрана программы.

4.3 Выводы

Программа работоспособна и оптимизирована с использованием ранее выбранных технологий. Анализ показал, что математические операции следует исполнять на более мощном аппаратном обеспечении, чем то, на котором производилось исследование.

ЗАКЛЮЧЕНИЕ

Цель работы достигнута - разработана программа моделирования движения воды с использованием воксельной графики. В результате проектирования использованы следующие методы:

— симуляция воды аппроксимируется с помощью клеточного автомата;

— генерация поверхности производится с помощью шума Перлина;

— для рендеринга вокселов используется метод реконструкции поверхностей с оптимизированной версией алгоритма марширующих кубов;

— для закраски поверхности использован метод Ламберта.

Реализованное в работе программное обеспечение соответствует указанным требованиям и работоспособно.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Тирский Г. А. Большая российская энциклопедия. Том 21. — Большая российская энциклопедия, 2012.

2. Bautembach Dennis. Animated Sparse Voxel Octrees // UNIVERSITY OF HAMBURG Department of Informatics. — 2011.

3. Zadick Johanne, Kenwright Benjamin, Mitchell Kenny. Integrating Real-Time Fluid Simulation with a Voxel Engine // The Computer Games Journal. — 2016.

4. The game of flow - cellular automaton-based fluid simulation for realtime interaction / Christian Heintz, Moritz Grunwald, Sarah Edenhofer et al. — 2017. — 11. — P. 1–2.

5. Premoze S., Ashikhmin M. Rendering Natural Waters. — Hong Kong, 2000. — P. 22–30.

6. Magnetic Resonance Imaging: Physical Principles and Sequence Design / Robert W. Brown, Y.-C. Norman Cheng, E. Mark Haacke et al. — John Wiley Sons, 2014. — P. 976.

7. VanderHart Luke, Neufeld Ryan. Clojure Cookbook: Recipes for Functional Programming. — O'Reilly Media, Inc., 2014.

8. Lombard Yann. Realistic Natural Effect Rendering: Water I. — 2004. — Access mode: `https://www.gamedev.net/articles/programming/graphics/realistic-natural-effect-rendering-water-i-r2138/`.

9. Wong Jamie. Ray marching and signed distance functions. — 2016. — 7. — Access mode: `http://jamie-wong.com/2016/07/15/ray-marching-signed-distance-functions/`.

10. Гаджиев А. М. Курс лекций по дисциплине "Компьютерная геометрия и графика". — 2004. — Режим доступа: `http://eor.dgu.ru/`.

11. NVIDIA. NVIDIA® GVDB Voxels. — 2018. — Access mode: https://developer.nvidia.com/gvdb.

12. Ash Michael. Simulation and Visualization of a 3D Fluid : Master's thesis / Michael Ash ; Université d'Orléans. — 2005.

13. Мякишев Г. Я. Кинетическое уравнение Больцмана. — Режим доступа: https://www.booksite.ru/fulltext/1/001/008/061/212.htm.

14. Баранов Василий. Моделирование гидродинамики: Lattice Boltzmann Method. — 2013. — Режим доступа: https://habr.com/ru/post/190552/.

15. Асламова В. С., Колмогоров А. Г., Сумарокова Н. Н. Вычислительная математика. Часть вторая: Учебное пособие для студентов дневного и заочного обучения технических и химико-технологических специальностей. — Ангарская государственная техническая академия, 2005. — С. 94.

16. Bate Matthew R., Bonnell Ian A., Bromm Volker. The Formation of Stars and Brown Dwarfs and the Truncation of Protoplanetary Discs in a Star Cluster. — 2002. — Access mode: https://www.ukaff.ac.uk/starcluster/.

17. Lacroute Philippe, Levoy Marc. Fast Volume Rendering Using a Shear-Warp Factorization of Viewing Transformation // Computer Graphics Proceedings, Annual Conference Series. — 1994.

18. Anderson Ben. An Implementation of the Marching Cubes Algorithm. — 2005. — Access mode: http://www.cs.carleton.edu/cs_comps/0405/shape/marching_cubes.html.

19. Bansall Sahil. Z-Buffer or Depth-Buffer method. — 2017. — Access mode: https://www.geeksforgeeks.org/z-buffer-depth-buffer-method/.

20. Hickey Rich. The Clojure Programming Language. — Access mode: https://clojure.org/.

21. Hickey Rich. Programming at the REPL: Introduction. — Access mode: https://clojure.org/guides/repl/introduction.

22. Hagelberg Phil. Leiningen. — Access mode: https://leiningen.org/.

ПРИЛОЖЕНИЕ А

РИСУНКИ

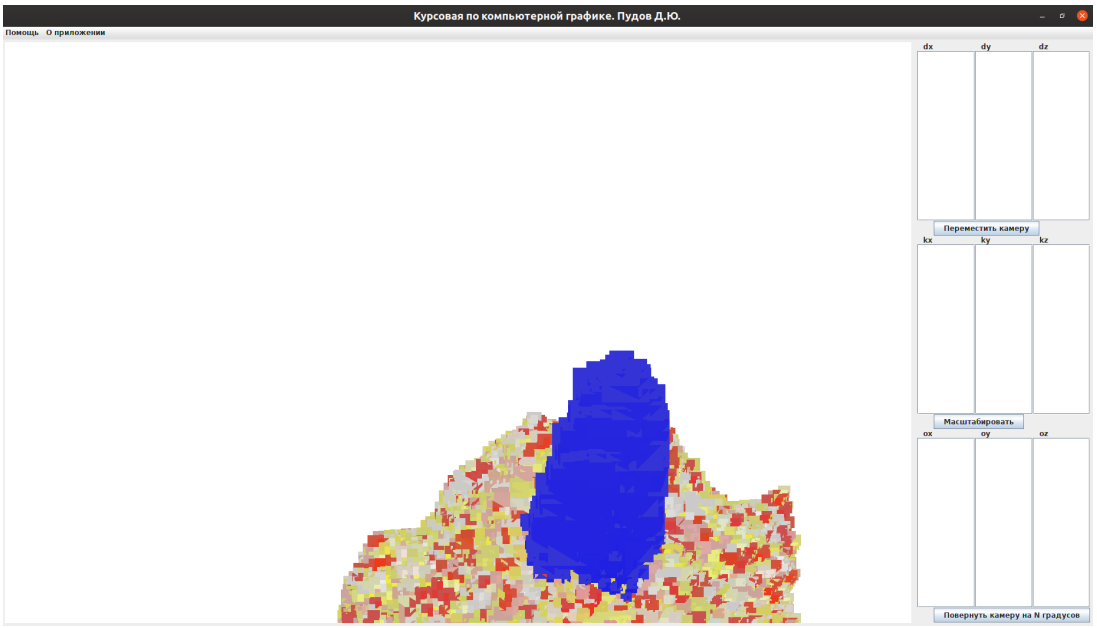

Рисунок А.1 — Снимок основного экрана программы