# Содержание

# Введение

В современном мире компьютерная графика вышла на совершенно новый уровень, что связано с ее высокой востребованностью в области игр и фильмов [1]. Методы отрисовки все время развиваются, появляются новые способы. Главным требованием является, в первую очередь, является реалистичность изображения. Множество исследований физических явлений происходит постоянно, чтобы с максимальной точностью смоделировать тот или иной объект. Но чем выше точность, тем выше сложность разрабатываемых алгоритмов, что часто приводит к высоким затратам по времени и памяти.

Одной из тем моделирования является моделирование жидкости [2]. Имеется огромная потребность в качественной и эффективной отрисовке океанов, рек, ручейков и даже луж. В данном курсовом проекте речь пойдет о моделировании водопада, с учетом аэрозольных облаков, которые возникают у подножия самого водопада при падении воды.

**Цель работы** – разработать программное обеспечение для визуализации геометрической модели водопада. Для выполнения поставленной цели необходимо решить следующие задачи:

- проанализировать методы и алгоритмы, моделирующие водопады;
- определить алгоритм, который наиболее эффективно справляется с поставленной задачей;
- реализовать алгоритм;
- разработать структуру классов проекта;
- провести эксперимент по замеру производительности полученного программного обеспечения.

## 1.2 Методы визуализации водопадов

Водопад представляет собой водяной поток с частицами брызг. За все время моделирования текучей воды было разработано несколько основных методов, конкретнее о которых речь и пойдет дальше.

### 1.2.1 Метод, основанный на уравнении Навье-Стокса

*Уравнение Навье-Стокса* [3] является системой дифференциальных уравнений в частных производных, которое описывает движение вязких ньютоновских жидкостей, которое до сих пор не имеет решения в общем виде.

$$\frac{d\vec{v}}{dt} = -(\vec{v}\bigtriangledown)\vec{v} + \nu \bigtriangleup \vec{v} - \frac{1}{\rho} \bigtriangledown p + \vec{f}, \tag{1.1}$$

где $\bigtriangledown$ – оператор набла, $\bigtriangleup$ – векторный оператор Лапласа, $t$ – время, $\nu$ – коэффициент кинематической вязкости, $\rho$ – плотность, $p$ – давление, $\vec{v}$ – векторное поле скорости, $\vec{f}$ – векторное поле массовых сил.

Уравнение часто используется для математического моделирования сложных моделей природных явлений. При разработке алгоритмов используются частные решения.

Методы, которые строятся на основе этого уравнения, являются довольно трудно реализуемы за счет большого количества сложной математики, с которой приходится работать, а также множества вычислений, которые приходится делать компьютеру при обработке всех формул, что является крайне неэффективным. По этим причинам методы на основе уравнения Навье-Стокса рассматриваться подробно не будут. [4]

### 1.2.2 Метод, основанный на частицах

Метод частиц заключается в использовании трилинейной интерполяции для каждой частицы, чтобы определить ее скорость движения. И каждая частица перемещается в соответствии с определенным инерциальным физическим уравнением. Данный подход имеет довольно небольшие затраты по

компьютерным ресурсам, а также большое количество частиц позволяет достичь максимальной точности изображения. Но от количества частиц зависит нагрузка на компьютер. Главным минусом является невозможность точно определить границы, в которых должен находиться водяной поток, что приводит к размещению дополнительных проверок, а также нет возможности создавать извивающиеся водяные потоки.

Наиболее распространённые методы с частицами представлены далее.

### Метод на основе понятия о полу-Лагранже

Метод на основе понятия о полу-Лагранже, который объединяется с новым подходом расчета жидкости вокруг объектов, позволяет эффективно решать уравнения движения жидкости, сохраняя при этом достаточно деталей, чтобы получить реалистичное изображение. Высококачественная поверхность получается из результирующего поля скоростей с использованием новой адаптивной техники для создания неявной поверхности [5].

### Метод, использующий диаграммы Вороного

Ключевым компонентом этого алгоритма является аппроксимация геометрии пены путем обработки частиц пузырьков как участков взвешенной диаграммы Вороного. Информация о связности, предоставляемая диаграммой Вороного, позволяет нам точно моделировать различные эффекты взаимодействия между пузырьками.

Пусть $P$ – точка, заданная в $R^3$. Тогда для любой точки $p$, принадлежащей $P$, ячейка Вороного $V_p$ точки $p$ определяется как место точки в $R^3$, имеющих $p$ в качестве ближайшего соседа в $P$:

$$V_p = \{x \in R^3, \forall q \in P : |x - p| < |x - q|\} \tag{1.2}$$

Каждая ячейка Вороного – выпуклая, а граница состоит из выпуклых граней меньшего размера. Совокупность ячеек Вороного и их граней образует комплекс ячеек в $R^3$, что и называется диаграммой Вороного.

Используя ячейки Вороного и веса, также можно явно решить проблему потери объема при моделировании пены, что является общей проблемой во многих подходах. Может быть встроен в симуляторы жидкости, в частности,

водопада [6].

### Метод, использующий уравнения движения частиц по криволинейной траектории

При моделировании каждая частица считается независимой и для нее вычисляется ее место в соответствии с физическим уравнением. Сами частицы случайным образом распределяются по линии течения воды.

В системе частиц каждая отдельно взятая частица будет рассматриваться в независимости от остальных частиц. В качестве приоритетного подхода к реализации модели частицы, падающей с вершины водопада, используется подход, который основан на криволинейном равноускоренном движении. Данный подход позволяет, используя известные физические законы для движения частицы, получить качественный результат.

**Для моделирования потока воды** в качестве основного уравнения берется закон равноускоренного движения:

$$y = y_0 + v_0 t + \frac{at^2}{2}, \tag{1.3}$$

где $y_0$ – начальное положение частицы, $v_0$ – начальная скорость движения частицы, $a$ – ускорение движения частицы, $t$ – время.

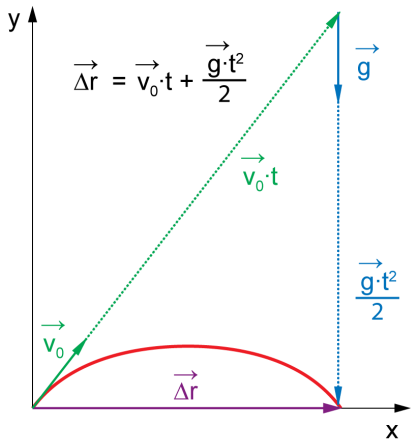


Рисунок 1.2 – Равноускоренное движение

При этом скорость частицы при равноускоренном движении вычисляется так:

$$v = v_0 + at, \tag{1.4}$$

где, $a = const$.

Само ускорение можно вычислить подобным образом:

$$a = \frac{dv}{dt} \tag{1.5}$$

Тогда основной закон равноускоренного движения можно записать так:

$$y = y_0 + v_0 t + \frac{vt}{2}, \tag{1.6}$$

Также нужно рассмотреть нахождение скорости, в зависимости направления, которое было задано частице. Вектор направления (D) в каждый момент времени вычисляется, путем сложения вектора направления движения и вектора гравитации. При этом получается новый вектор направления – результирующий вектор, тогда скорость будет вычисляться так:

$$\vec{v} = v * \vec{R}, \tag{1.7}$$

где $\vec{v}$ – вектор скорости частицы, $v$ – скалярная скорость частицы, а $\vec{R}$ – результирующая вектора направления и вектора гравитации.

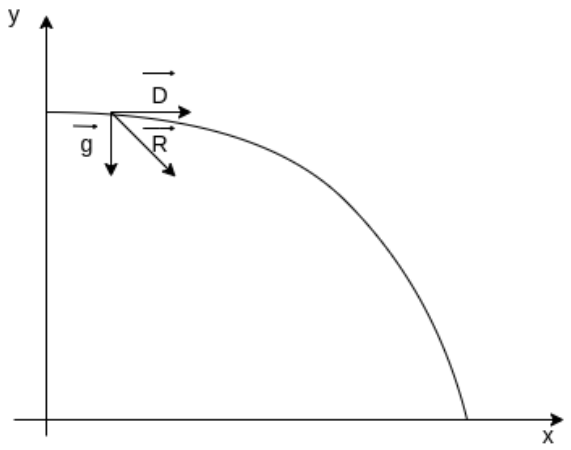


Рисунок 1.3 – Результирующий вектор

**Водяной пар** моделируется определенным образом. Поскольку при ударе воды о водяную поверхность образуется пар, то его также необходимо смоделировать. Полученная система частиц при реализации основного потока водопада позволяет получить довольно простую реализацию водяного пара.

При достижении водоема, частица отражается, теряя при этом 70% от скорости, и перекрашивается в белый цвет, обозначая тем самым, что она

превратилась в пар. При этом и изменяется направление ее движения, то в векторе направления координата по $z$ умножается на минус один.

**Водяные брызги** также можно получить из имеющегося водяного потока. Благодаря использованию системы частиц просто смоделировать частицы брызг, которые отходят от водопада. Поскольку каждая частица имеет определенное время жизни (так как частицы, которые прошли стадию водопада и пара должны быть удалены), она при скором достижении конца жизни (примерно на 100 моментов времени меньше, чем максимальное время жизни частицы) перекрашивается в белый, чтобы показать, что она стала брызгом, уменьшается на 30% скорость, а также у нее изменяются значения вектора направления, умножаясь на определенные коэффициенты, что в данном случае для $y$ координаты равно 1.014.

### 1.2.3 Сеточные методы

Сетки часто используются при моделировании жидкостей. В свою очередь, сама сетка – регулярная кубическая, в каждой ячейке которой содержится информация о точке поверхности, которая задает саму картину. Данный метод при большом количестве ячеек вызывает большое потребление памяти, хотя чем больше ячеек, тем выше качество получаемого изображения. Но также этот метод имеет и свои преимущества. Во-первых, визуализировать результат симуляции проще. Во-вторых, точно известны границы желаемой области, что приводит к возможности встроить данный водяной поток в нужное место.

Наиболее распространённые методы с сеткой представлены ниже.

#### Метод моделирования пузырька воздуха в воде

Данный метод представляет собой физически обоснованный метод расчета размеров пузырьков воздуха, основанный на скорости замерзания и давлении. В отличие от большинства методов, этот метод может представлять множество мелких пузырьков, которые не могут быть представлены с помощью решений имитационных сеток [7];

### Метод, использующий уравнение Эйлера

В данном методе используется уравнение Эйлера движения невязкой частицы:

$$\frac{du}{dt} = -(u \cdot \bigtriangledown)u + \frac{f}{\rho} - \frac{\bigtriangledown p}{\rho}, \tag{1.8}$$

при этом учитывается ограничение несжимаемости:

$$\bigtriangledown \cdot u = 0, \tag{1.9}$$

где $u = [u, v, w]^T$ – поле скорости жидкости, $p$ – давление, $t$ – время, $\rho$ – плотность жидкости, а $f$ – поле внешних сил.

Благодаря новому подходу можно моделировать крупномасштабные трехмерные жидкости. Здесь используется гибридное представление сетки, состоящее из правильных кубических ячеек поверх слоя высоких ячеек. Также благодаря оптимизации в представлении сетки была предложена реализация ГПУ решателя для жидкости [8].

## 1.2.4 Комбинированный метод

Методы, основанные на частицах и на сетках, было предложено объединить в один. Совместное их использование помогает одновременно с потоком воды моделировать и другие виды водяных потоков. При создании полноценной системы данный метод является предпочтительным.

Наиболее распространённые комбинированные методы представлены ниже.

### Метод, при котором моделируются пузырьки воздуха и пена вместе

Метод использует уравнение гидродинамики сглаженных частиц (SPH). В данном методе жидкость превращается в набор частиц $i$ с положением $x_i$ и скоростью $v_i$. Как правило, количество частиц $A_i$ аппроксимируется функцией сглаживания, которая интерполирует $A_i$, используя конечный набор точек выборки $j$, находящихся на расстоянии $h$. Этот набор точек выборки называется *окрестностью частицы*. Функция сглаживания определяется так:

$$A_i = \sum_j \frac{m_j}{\rho_j} A_j W(x_i - x_j h), \tag{1.10}$$

где $m_j$ – масса j, $\rho_j$ – ее плотность, а $W(x_i - x_j, h)$ – функция ядра (кубический сплайн).

Фазы моделирования пузырьков воздуха и пены рассматриваются отдельно. Также в этом методе применяется такие подходы, как использование силы сопротивления (для того, чтобы совместить оба этапа и при этом сильно не увеличить вычислительные нагрузки), функция насыщения (зависит от объема). При достижении поверхности пузырьки преобразуются в пену, а потом удаляются за указанное пользователем время [9];

**Метод, который напрямую использует подходы на основе сеток и частиц**

Области жидкости, которые не могут быть представлены полем полем высоты (например, разбивающиеся волны и брызги из-за ударов воды), то они представляются частицами и превращаются в брызги и пену. Сами частицы рассматриваются как простые невзаимодействующие точечные массы, которые обмениваются массой и моментом с жидкостью поля высоты [10].

### Вывод

В качестве приоритетного выбирается метод, основанный на частицах. Выбор заключается в том, что данный метод предоставляет возможность смоделировать как сам водопад, который представляет из себя большое количество частиц, так и различные брызги и пену, которая также будет являться частицей.

## 1.3 Формализация модели

Модель водопада будет задаваться такими характеристиками, как:

- высота – высота уступа, с которого будет падать вода. Число типа $float$ в промежутке от 3 до 30 (в метрах);
- ширина – по сути, ширина уступа, с которого течет вода. Число типа $float$ в промежутке от 1 до 10 (в метрах);
- угол падения – угол, который зависит от уступа, с которого падает водопад; число типа $float$;
- скорость водопада – число типа $float$;
- количество частиц – число типа $int$;
- размер частиц – число типа $int$.

Также частью сцены будет являться скала, которая будет задаваться геометрическим объектом – параллелепипедом. Со скалы должен падать водопад.

## 1.4 Выбор метода рендера изображения

Рендеринг или отрисовка – термин в компьютерной графике, обозначающий процесс получения изображения по модели с помощью компьютерной программы.

Основным методом для генерации водопада была выбрана система частиц, которая показывает наилучшие результаты при большом количестве объектов. Программа должна работать быстро, чтобы был виден результат, похожий на поток воды. Для рендера системы частиц не подходит стандартная графическая библиотека, поскольку она не справится с нагрузкой, которая будет на нее возложена. Следовательно, необходимо выбрать API, которое позволит более гибко управлять данными, а также использовать графический ускоритель для рендера изображения. Основными API являются DirectX, Vulkan и OpenGL.

### 1.4.1 DirectX

**DirectX** – это последняя версия собственного API компьютерной графики от Microsoft, используемого для платформ Windows и Xbox. Он нацелен на создание менее сложного драйвера и API, более близкого к архитектуре современных графических процессоров. DirectX фокусируется на рендеринге в реальном времени, поэтому он предназначен для разработчиков игр и систем автоматизированного проектирования (CAD). Поскольку это отраслевой стандарт API компьютерной графики, можно ожидать, что почти все совместимые аппаратные средства будут иметь его надежную поддержку, и он станет стандартом для коммерческих проектов.

Этапы рендеринга растровой графики с помощью DirectX очень похожи на этапы других современных графических API.

1. *Инициализация API* — создаются:

    - фабрика – точка входа в DirectX API;
    - адаптер – предоставляет информацию о физических характеристиках данного устройства DirectX;
    - устройство – основная точка входа в DirectX API, предоставляющая доступ к внутренним частям API. Это ключ к важным структурам данных, таким как конвейеры, шейдеры, состояние рендеринга;
    - очередь команд – позволяет отправлять группы вызовов отрисовки, известных как списки команд, для выполнения по порядку, что позволяет графическому процессору оставаться занятым и оптимизировать скорость его работы;
    - распределитель команд – используется для создания списка команд, то есть структуры данных, в которой выполняются вызовы отрисовки к GPU.

2. *Инициализация ресурсов* — создаются:

    - буфер вершин – хранит информацию о каждой вершине, доступную в виде атрибутов в вершинном шейдере;

- индексный буфер – содержит индивидуальные индексы каждых треугольника/линии/точки, которые нужно нарисовать;
- однородный буфер – описывает данные, которые будут посылаться во время отрисовки к стадиям шейдеров (например, цвет).

3. *Визуализация* — обновление однородных данных, добавление команд в очередь и ожидание следующего кадра.

**Преимущества:**

- высокое качество изображения;
- быстрый рендер избораженния.

**Недостатки:**

- новые версии только для самого современного оборудования (не поддерживается старыми видеокартами);
- используется только на Windows.

### 1.4.2 OpenGL

OpenGL – в большинстве случаев рассматривается как API, предоставляющий большой набор функций, которые можно использовать для управления графикой и изображениями. OpenGL является спецификацией, разработанной и поддерживаемой Khronos Group.

Спецификация OpenGL определяет, каким должен быть результат/вывод каждой функции, и как она должна выполняться. А вот реализация этой спецификации уже зависит от конкретных разработчиков.

Если не ограничиваться аппаратурой, а рассмотреть поддержку графической обработки для современных систем в целом, то видно, что она базируется на понятии конвейера: графические данные проходят последовательно несколько этапов обработки – выходные данные одного этапа сразу передаются на вход следующего. Абстрагируясь от связанных с конкретными реализациями деталей, можно рассмотреть универсальный графический конвейер и выделить в нем 5 этапов.

1. Этап генерации (G) – создание и модификация прикладных структур данных.

2. Этап обхода (T) прикладных структур данных и порождение соответствующих графических данных.

3. Этап преобразования (X), на котором графические данные из системы координат объекта преобразуются в систему координат наблюдателя, выполняется расчет освещенности, отсечение преобразованных данных, а затем проецирование результата в пространство окна.

4. На этапе растеризации (R) создаются и записываются в буфер кадра дискретные образы примитивов: точки, отрезки и полигоны. Буфер кадра – это банк памяти, предназначенный для хранения массива пикселей изображения. На этом этапе для всех вершин геометрических объектов вычисляется закраска, производится наложение определенных участков текстуры, а также выполняются пиксельные операции, такие, например, как сравнение по глубине.

5. На этапе вывода (D) происходит сканирование буфера кадра и вывод изображения на экран дисплея.

**Преимущества:**

- высокая производительность работы;
- гибкая структура, которая позволяет изменять параметры выводимых объектов;
- кроссплатформенность (используется на Linux, MacOS, Windows);
- библиотеки для работы с API на различных языках программирования.

**Недостатки:**

- сложная работа с новыми возможностями GPU;
- не имеет возможности для работы с мышью и клавиатурой (нужно отдельное API).

### 1.4.3 Vulkan

Vulkan — кроссплатформенный API для 2D- и 3D-графики, представленный Khronos Group.

Vulkan API изначально был известен как «новое поколение OpenGL» или просто «glNext», но после анонса компания отказалась от этих названий в пользу названия Vulkan. Как и OpenGL, Vulkan позволяет с высокой производительностью отображать в реальном времени различные приложения с 3D-графикой, такие как игры или интерактивные книги на всех платформах, а также обеспечивает более высокую производительность и меньшую нагрузку на процессор.

Перед получением изображения на экран, Vulkan выполняет ряд следующих действий.

1. **Подготовка**. Нужно сообщить Vulkan, какие буферы будут использоваться во время рендеринга. Необходимо указать, сколько будет буферов цвета, буферов глубины. Также нужно указать, как должно обрабатываться содержимое буферов во время рендеринга. Вся эта информация обернута в объект прохода рендера.

2. **Настройка буферов**. Настраиваются буферы глубины и цвета, в которые загружаются данные. При конфигурации используются два основных метода, для каждого буфера: *loadOp* и *storeOp*. Для *loadOp* возможны следующие значения:

   - $VK\_ATTACHMENT\_LOAD\_OP\_LOAD$ – буфер будет содержать те данные, которые были помещены в него до этого прохода (например, во время предыдущего прохода);
   - $VK\_ATTACHMENT\_LOAD\_OP\_CLEAR$ – буфер очищается в начале прохода рендера;
   - $VK\_ATTACHMENT\_LOAD\_OP\_DONT\_CARE$ – содержимое буфера не определено.

   Для *storeOp* возможны два значения:

- *VK_ATTACHMENT_STORE_OP_STORE* – содержимое буфера сохраняется в память для дальнейшего использования;
- *VK_ATTACHMENT_STORE_OP_DONT_CARE* – после рендеринга буфер больше не используется, и его содержимое не имеет значения.

3. **Подпроходы**. Один проход рендера может состоять из множества подпроходов. Подпроходы – это последовательные операции рендеринга, зависящие от содержимого фреймбуферов в предыдущих проходах. К ним относятся, например, эффекты постобработки, применяемые друг за другом. Если объединить их в один проход рендера, Vulkan сможет перегруппировать операции для лучшего сохранения пропускной способности памяти и большей производительности. Каждый подпроход ссылается на один или несколько буферов.
4. **Проход рендера**. Вывод изображения на экран дисплея.

**Преимущества:**

- высокая производительность работы;
- открытый код;
- кроссплатформенность (используется на Linux, MacOS, Windows).

**Недостатки:**

- работа на глубоком уровнем GPU;
- не имеет библиотек, позволяющих полноценно использовать Vulkan на другом языке.

## Вывод

Проанализировав различные API для рендера изображения на экран, был выбран OpenGL. Он отличается кроссплатформенностью, что дает ему выигрыш над DirectX, потому что DirectX используется исключительно для операционных систем Windows. Также Vulkan не имеет библиотек для языка

# Вывод

В данном разделе были формально описаны все методы по визуализации текучей воды, с помощью которых можно получить реализацию водопада, а также методы рендера изображения. В качестве алгоритма визуализации водопада предпочтение отдается методу, который реализует подход, основанный на системе частиц. Также OpenGL был выбран в качестве метода рендера изображения.

# 2 Конструкторская часть

В данном разделе будут представлены требования к программному обеспечению, а также схемы алгоритмов, выбранных для решения задачи.

## 2.1 Требования к программному обеспечению

Программа должна предоставлять доступ к функционалу:

- изменение параметров модели водопада в активном режиме: высота водопада, угол падения воды, скорость водяного потока;
- изменение параметров частиц, из которых состоит водопад, в активном режиме: количество частиц, размер частиц;
- включение и выключение работы модели водопада;
- вращение, перемещение и масштабирование модели.

Требования, которые предъявляются к программе:

- время отклика программы должно быть менее 1 секунды для корректной работы в интерактивном режиме;
- программа должна корректно реагировать на любое действие пользователя.

## 2.2 Разработка алгоритмов

В данном разделе будут представлены схемы реализации выбранных алгоритмов.

### 2.2.1 Система частиц для реализации водопада

На рисунке 2.1 показана схема алгоритма реализации движения частицы и возможные этапы ее превращения в пар и брызг.

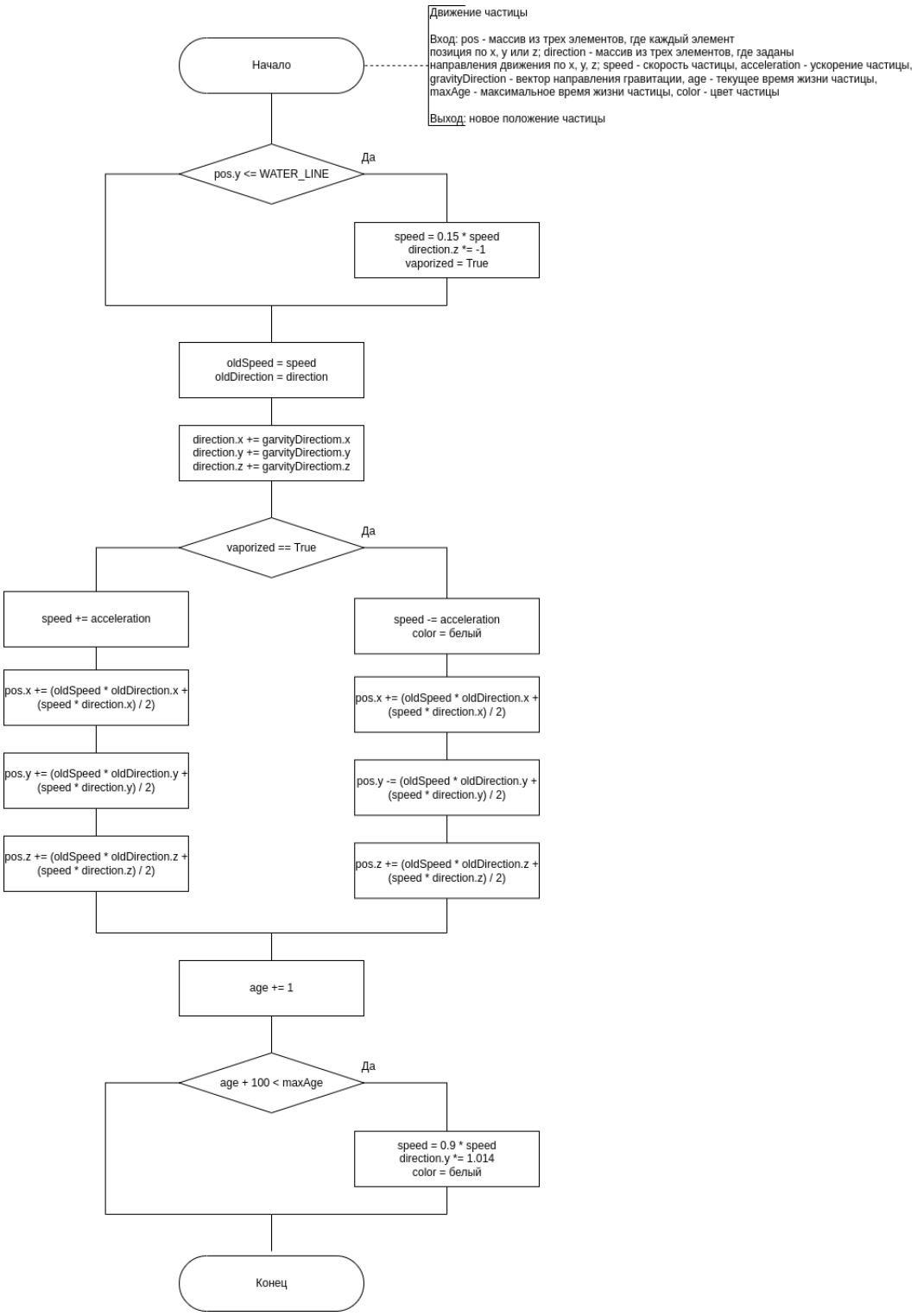


Рисунок 2.1 – Схема алгоритма движения частицы водопада

### 2.2.2 Отрисовка изображения

На рисунке 2.2 показана схема алгоритма отрисовки части сцены, отвечающей за водопад, а на рисунке 2.3 – схема алгоритма для отрисовки остальных объектов сцены (скалы и водного полотна).

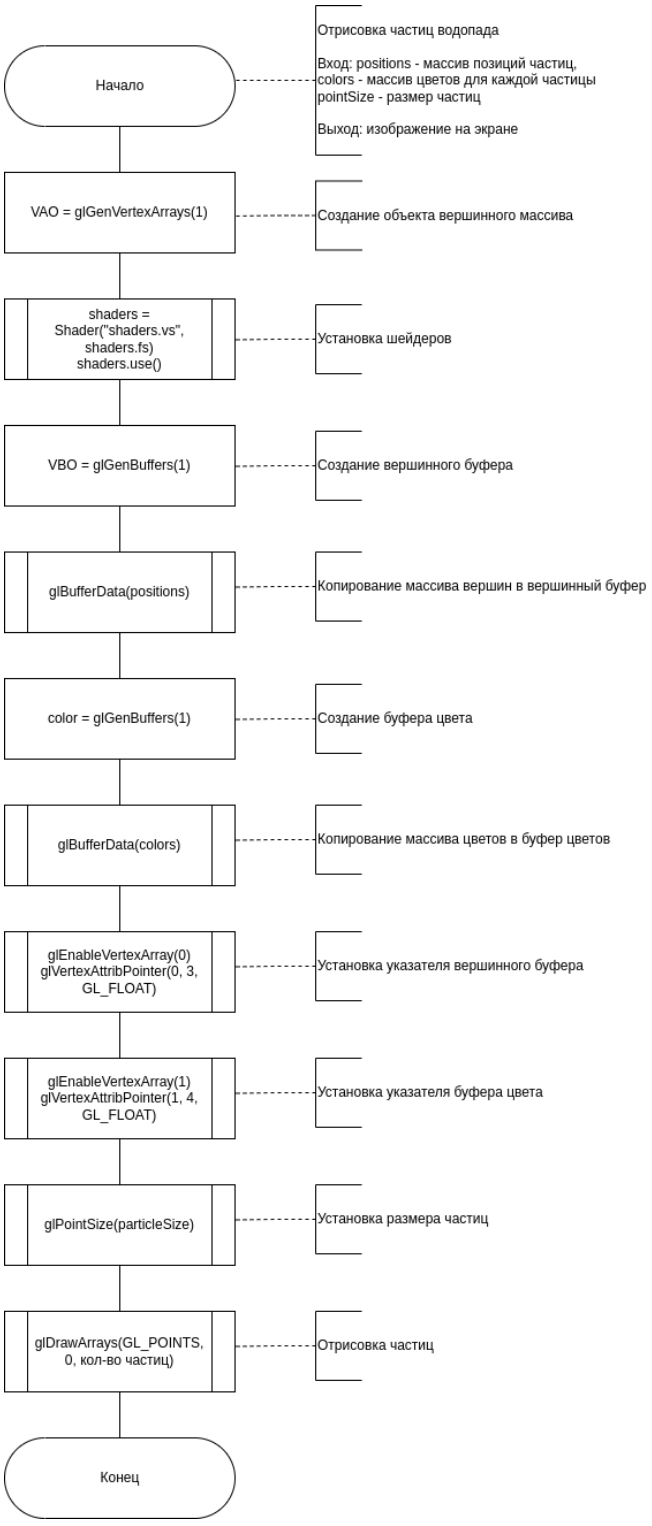


Рисунок 2.2 – Схема алгоритма отрисовки водопада

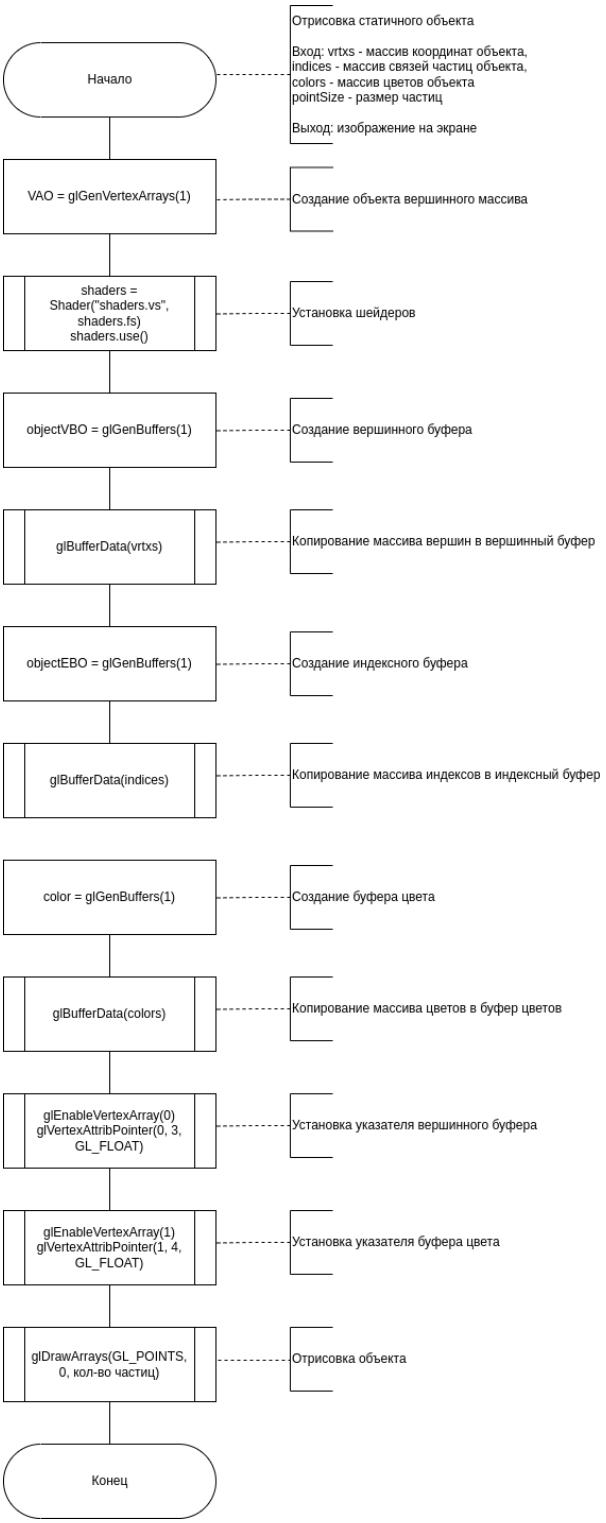


Рисунок 2.3 – Схема алгоритма отрисовки статичного объекта (скалы)

## 2.3 Описание структуры программы

На рисунке 2.4 показана структура реализуемых классов.

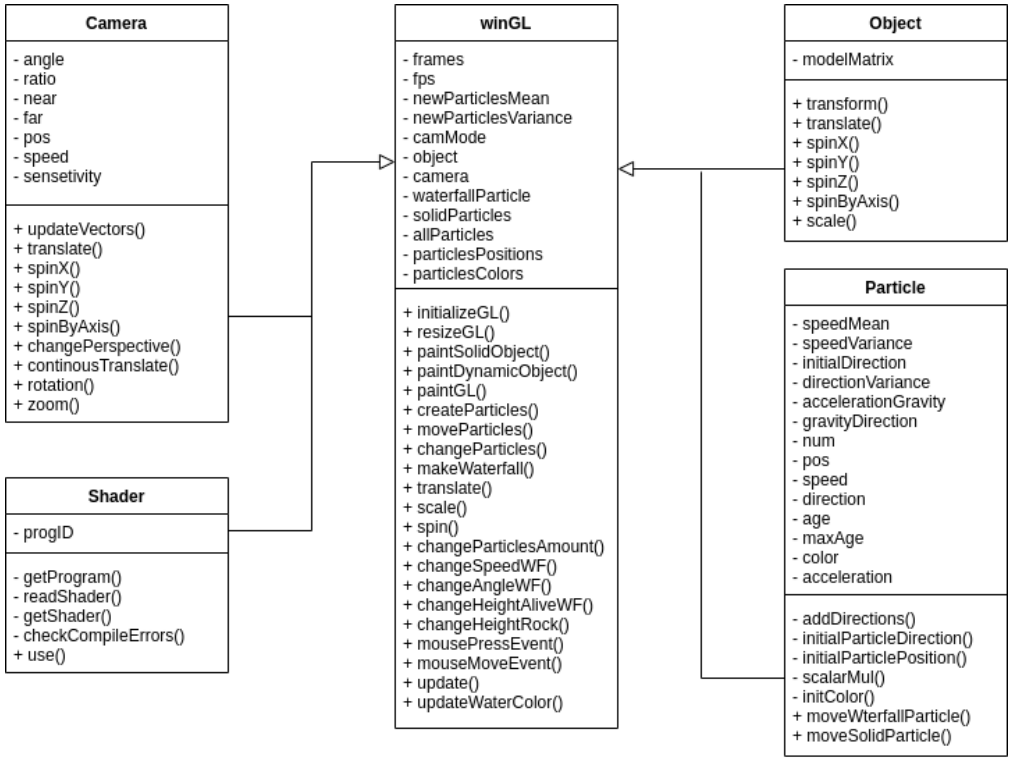


Рисунок 2.4 – Схема классов программы

Описание реализуемых классов:

- **Camera** – класс для работы с камерой. Хранит позицию камеры, угол ее наклона, скорость перемещения для динамической смены положения камеры;
- **Shader** – класс, который подключает шейдеры к приложению. Хранит данные шейдеров для передачи цвета объектов;
- **Object** – класс, который владеет информацией о всех объектах сцены;
- **winGL** – класс для отрисовки объектов сцены. Хранит массив частиц для водопада, хранит массив цветов для водопада, а также данные для отрисовки статичных объектов;
- **Particle** – класс, который описывает одну частицу водопада. Хранит ее положение, направление движения, цвет, скорость, время жизни и максимальное время жизни.

## 2.4 Используемые типы и структуры данных

При реализации программного обеспечения были использованы следующие типы и структуры данных:

- параметры водопада: высота, ширина, скорость, размер частиц – числа типа $float$;
- параметры водопада: количество частиц – число типа $int$
- точка – массив координат по осям X, Y, Z;
- объект – массив точек с координатами вершин, также массив связей между номерами вершин;
- водопад – массив частиц (объектов класса $Particle$).

# Вывод

В данном разделе были рассмотрены требования, которые выдвигаются программному продукту, схемы алгоритмов, а также типы и структуры данных, которые были использованы при реализации ПО.

# 3 Технологическая часть

В данном разделе будут рассмотрены средства разработки программного обеспечения, детали реализации, а также диаграмма классов.

## 3.1 Средства реализации

При написании программного продукта был выбран язык *Python* [11]. Это обусловлено следующими факторами:

- объектно-ориентированный язык, что позволяет использовать структуру классов;
- имеются необходимые библиотеки для реализации поставленной задачи;
- существует много учебной литературы.

В качестве разработки интерфейса был выбран *Qt Designer* [12]. Он позволяет создать качественный интерфейс, так как имеет встроенный редактор выводимого окна.

В качестве среды разработки был выбран *Visual Studio Code* [13]. Данное приложение имеет большое количество плагинов для работы с кодом.

## 3.2 Реализация алгоритмов

В листинге 3.1 представлен алгоритм перемещения частицы водопада за один кадр.

Листинг 3.1 – Алгоритм перемещения частицы водопада за один кадр

```
1 def moveWaterfallParticle(self):
2     if (self.pos[1] <= WATER_LINE):
3         self.speed = BOUNCE_COEF * self.speed
4         self.direction[2] *= -1
5         self.vaporized = True
6 
7     oldSpeed = self.speed
```

```
8       oldDirection = deepcopy(self.direction)
9
10      self.direction[0] += self.gravityDirection[0]
11      self.direction[1] += self.gravityDirection[1]
12      self.direction[2] += self.gravityDirection[2]
13
14      if (self.vaporized):
15          self.speed -= self.acceleration
16
17          self.pos[0] = self.pos[0] + oldSpeed * oldDirection[0] +
              (self.speed * self.direction[0]) / 2
18          self.pos[1] = self.pos[1] - (oldSpeed * oldDirection[1] +
              (self.speed * self.direction[1]) / 2)
19          self.pos[2] = self.pos[2] - (oldSpeed * oldDirection[2] +
              (self.speed * self.direction[2]) / 2)
20      else:
21          self.speed += self.acceleration
22
23          self.pos[0] = self.pos[0] + oldSpeed * oldDirection[0] +
              (self.speed * self.direction[0]) / 2
24          self.pos[1] = self.pos[1] + oldSpeed * oldDirection[1] +
              (self.speed * self.direction[1]) / 2
25          self.pos[2] = self.pos[2] + oldSpeed * oldDirection[2] +
              (self.speed * self.direction[2]) / 2
26
27      self.age += 1
28
29      if (self.age + 100 > self.maxAge):
30          self.speed *= 0.9
31          self.direction[2] *= 1.014
32          self.color = glm.vec4(1, 1, 1, 1)
33
34      return self
```

# Вывод

В данном разделе были рассмотрены средства, с помощью которых было реализовано ПО, описана структура классов проекта, а также представлен листинг алгоритма перемещения частицы водопада за кадр.

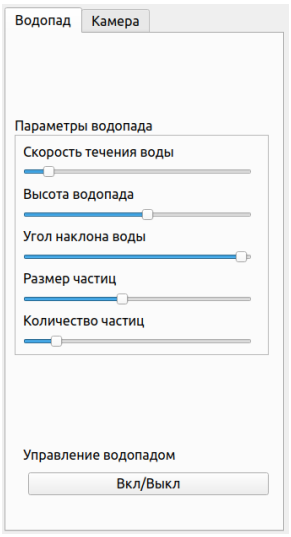


Рисунок 4.4 – Окно управления водопадом

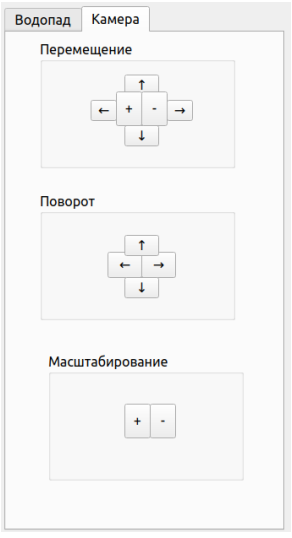


Рисунок 4.5 – Окно управления камерой

## 4.2 Постановка эксперимента

### 4.2.1 Цель эксперимента

Целью эксперимента является проведение тестирования производительности при создании сцен различной нагруженности. Нагрузка будет меняться в зависимости от количества частиц, из которых состоит водопад.

Оцениваться производительность будет мерой количества кадров в секунду (Frames Per Second, FPS, к/с), которое получается при работе приложения при данной загруженности.

### 4.2.2 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование представлены далее:

- операционная система: Ubuntu 20.04.3 [14] Linux [15] x86_64;
- память: 8 GiB;
- процессор: Intel® Core™ i5-7300HQ CPU @ 2.50GHz [16];
- видеокарта: NVIDIA® GeForce® GTX 1050Ti with 4 GB GDDR5 Dedicated VRAM [17].

При тестировании ноутбук был включен в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, а также системой тестирования.

### 4.2.3 Результаты эксперимента

Результаты эксперимента приведены в таблице 4.1. Также на рисунке 4.6 представлен график изменения FPS в зависимости от количества частиц.

Таблица 4.1 – Зависимость производительности от количества частиц

| **Частиц, штук** | **Производительность, к/с** |
|---|---|
| 500 | 210 |
| 1000 | 120 |
| 2000 | 80 |
| 3000 | 50 |
| 4000 | 34 |
| 5000 | 27 |
| 6000 | 21 |
| 7000 | 18 |
| 8000 | 16 |
| 9000 | 14 |
| 10000 | 12 |
| 11000 | 11 |
| 12000 | 10 |
| 13000 | 9 |
| 14000 | 8 |
| 15000 | 7 |

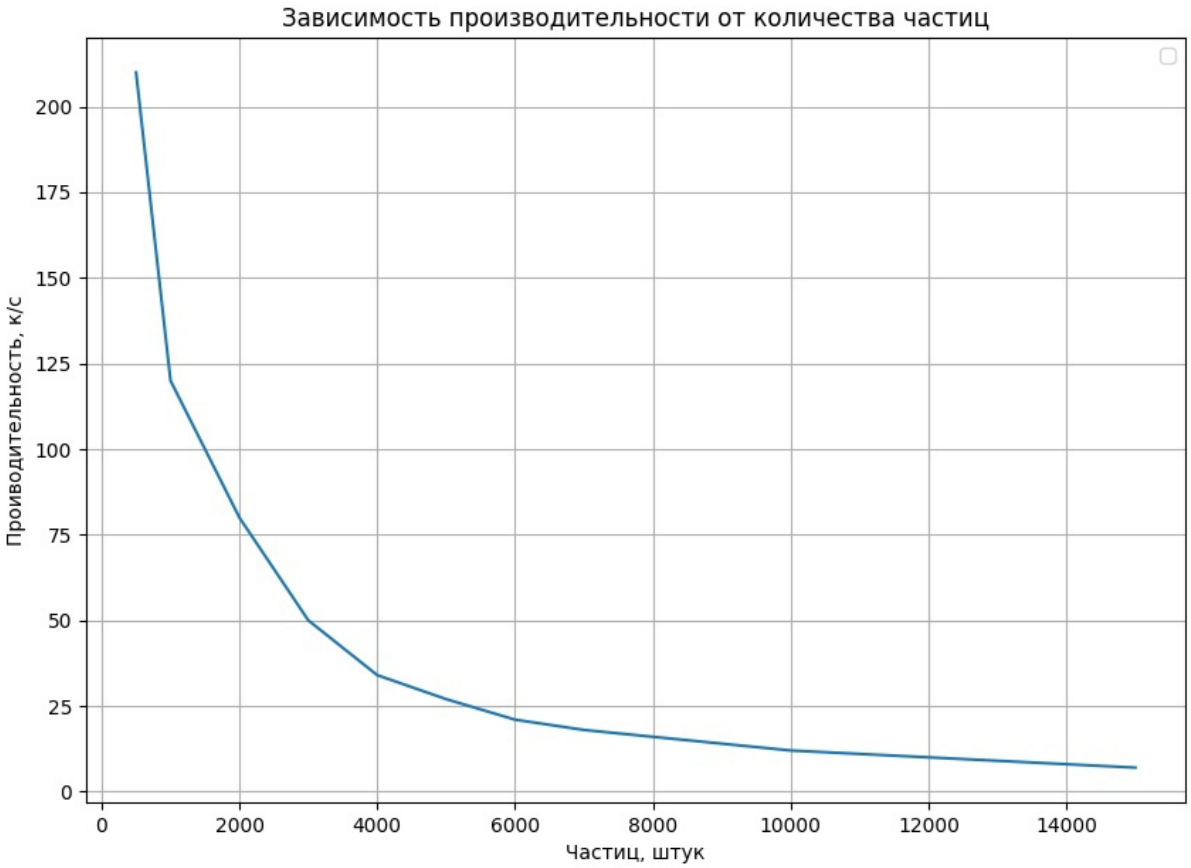


Рисунок 4.6 – Зависимость к/с от количества частиц в водопаде

### Вывод эксперимента

Как видно из результатов, количество кадров в секунду уменьшается экспоненциально при линейном увеличении количества частиц в водопаде. Рендер большого количества частиц, несмотря на хорошее программное обеспечение компьютера, является трудной задачей: при 1000 частиц программа выдает 120 FPS, в то время как при уже при 10000 тысячах частиц получается 12 FPS.

Для комфортной работы человеку необходимо 24 FPS [18], что получается при 5000 тысячах частиц. При этом частиц достаточно, чтобы они вместе были похожи на водопад.

## Вывод

В данном разделе были рассмотрены примеры работы программного обеспечения, а также выяснено в результате эксперимента, что производительность программы (в FPS) падает по экспоненциальному закону при линейном увеличении количества частиц.

# Заключение

В ходе программного обеспечения было разработано программное обеспечение, которое реализует модель водопада по методу частиц. Полученное приложение позволяет изменять параметры водопада в интерактивном режиме (скорость течения воды, угол наклона падающей воды, высота водопада, размер частиц, количество частиц). В процессе выполнения данной работы были выполнены следующие задачи:

- рассмотрены методы реализации модели водопада;
- выбран алгоритм, который наиболее эффективно решает поставленную задачу;
- реализован выбранный алгоритм;
- разработана структура классов проекта;
- проведен эксперимент по замеру производительности полученного программного обеспечения.

В процессе исследования было выявлено, что производительность программы понижается экспоненциально при линейном увеличении количества частиц. При этом водопад визуально выглядит лучше при наибольшем количестве частиц, которые вместе составляют единую структуру.

# Литература


[1] Li K. Water Simulating in Computer Graphics [Электронный ресурс] // Швеция, Linnaeus University. 2007.

[2] Brunner P. Modeling Surface Water-Groundwater Interaction with MODFLOW: Some Considerations [Электронный ресурс] // США, Groundwater. 2010.

[3] The Navier-Stokes Equations [Электронный ресурс]. Режим доступа: `https://www.iit.edu/sites/default/files/2021-02/navier_stokes.pdf` (дата обращения: 23.09.2021).

[4] Wang Y. Solving 3D incompressible Navier-Stokes equations on hybrid CPU/GPU systems [Электронный ресурс] // Париж, Universite Paris-Sud. 2014.

[5] Foster N. Practical Animation of Liquids [Электронный ресурс] // США, SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques. 2001.

[6] Busaryev O. Animating Bubble Interactions in a Liquid Foam [Электронный ресурс] // США, ACM Transactions on Graphics. 2012.

[7] Iwasaki K. Visual Simulation of Freezing Ice with Air Bubbless [Электронный ресурс] // Сингапур, SIGGRAPH Asia 2012 Technical Briefs. 2012.

[8] Chentanez N. Real-Time Eulerian Water Simulation Using a Restricted Tall Cell Grid [Электронный ресурс] // США, ACM Transactions on Graphics. 2011.

[9] Ihmsen M. ANIMATION OF AIR BUBBLES WITH SPH [Электронный ресурс] // Фрайбург, Computer Science Department – University of Freiburg. 2011.

[10] Chentanez N. Real-time Simulation of Large Bodies of Waterwith Small Scale Details [Электронный ресурс] // Мадрид, ACM SIGGRAPH Symposium on Computer Animation. 2010.

[11] Welcome to Python [Электронный ресурс]. Режим доступа: https://www.python.org (дата обращения: 23.11.2021).

[12] Qt Designer Manual [Электронный ресурс]. Режим доступа: https://doc.qt.io/qt-5/qtdesigner-manual.html (дата обращения: 23.11.2021).

[13] Visual Studio Code [Электронный ресурс]. Режим доступа: https://code.visualstudio.com/docs (дата обращения: 23.11.2021).

[14] Ubuntu 20.04.3 LTS (Focal Fossa) [Электронный ресурс]. Режим доступа: https://releases.ubuntu.com/20.04/ (дата обращения: 27.11.2021).

[15] Linux [Электронный ресурс]. Режим доступа: https://www.linux.org/forums/#linux-tutorials.122 (дата обращения: 27.11.2021).

[16] Процессор Intel® Core™ i5-7300HQ [Электронный ресурс]. Режим доступа: https://ark.intel.com/content/www/ru/ru/ark/products/97456/intel-core-i5-7300hq-processor-6m-cache-up-to-3-50-ghz.html (дата обращения: 25.11.2021).

[17] GEFORCE GTX 1050 [Электронный ресурс]. Режим доступа: https://www.nvidia.com/en-in/geforce/products/10series/geforce-gtx-1050/ (дата обращения: 25.11.2021).

[18] Debattista K. Frame Rate vs Resolution: A Subjective Evaluation of Spatiotemporal Perceived Quality Under Varying Computational Budgets [Электронный ресурс] // Чичестер, Computer Graphics Forum. 2017.