МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт **математики, физики и информационных технологий**
Кафедра **«Прикладная математика и информатика»**

01.03.02 ПРИКЛАДНАЯ МАТЕМАТИКА И ИНФОРМАТИКА

СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ И КОМПЬЮТЕРНЫЕ ТЕХНОЛОГИИ

БАКАЛАВРСКАЯ РАБОТА

на тему «Разработка алгоритмов реального времени формирования видеоряда в нагруженных системах с использованием технологии CUDA»

Студент О.В. Ярыгин _____

Руководитель Т.Г. Султанов _____

Консультант по аннотации Н.В. Ященко _____

Допустить к защите

Заведующий кафедрой к.т.н., доцент А.В. Очеповский _____

«_____» _____ 2017 г.

Тольятти 2017

АННОТАЦИЯ

Тема данной выпускной квалификационной работы – «Разработка алгоритмов реального времени формирования видеоряда в нагруженных системах с использованием технологии CUDA».

Объектом исследования является процесс построения видеоряда. Предметом исследования являются параллельные алгоритмы высокопроизводительной визуализации видеорядов на основе заданной траектории движения камеры.

Актуальность исследования заключается в возрастающей востребованности высокопроизводительных вычислений, производимых для одновременного выполнения ресурсоёмких запросов множества клиентов.

Цель работы – разработка программного обеспечения для получения наибольшей производительности с точки зрения количества одновременно выполняемых запросов посредством разработки специальных алгоритмов и методов визуализации.

В настоящую выпускную квалификационную работу входит введение, три главы, итоговое заключение и список использованных источников.

В первой главе изучается теоретический материал по выбранной теме и текущий уровень развития отрасли, а также существующие программные средства, направленные на решение проблемы настоящей работы, с последующим формированием математической модели.

Во второй главе осуществляется практическая реализация сформулированной математической модели в виде программы на языке C++11, использующей технологию CUDA.

В третьей главе производится количественная оценка производительности и сравнение её с имеющимися решениями; а также формальное доказательство соответствия программы заявленным требованиям.

Заключение содержит основные выводы по проделанной работе и описывает итоги по выполненным задачам.

Объём выпускной квалификационной работы составляет 46 страниц, на которых размещены 19 рисунков и две таблицы, имеются 6 приложений. При написании выпускной квалификационной работы был использован 31 источник, 14 из которых на английском языке.

Ключевые слова: компьютерная графика, 3D, рендеринг, производительность, оптимизация, CUDA, shared computing, распределённые системы, распределённые вычисления, математическое моделирование, суперкомпьютинг, обработка видео, видеоряд.

ABSTRACT

The graduation work consists of an explanatory note on 46 pages, introduction, three parts including 19 figures, 2 tables, the list of 31 references including 14 foreign sources and 6 appendices. The title of the graduation work is “Development of Real-time Video Sequence Creation Algorithms for High Load Systems Using CUDA”.

The aim of the work is to develop software by means of special rendering algorithms and methods for a purpose of maximum performance gain of simultaneously executing queries.

The work is devoted to a problem of massive video sequences rendering using remote service, where the service is dedicated to process a large quantity (in order of thousands) of simultaneous requests in near real time manner.

The object of the graduation work is a process of video sequence creation. The subject of the graduation work is high-efficiency camera trajectory based parallel rendering algorithms.

In the first part a theoretical material is studied. Then, existing alternative software solutions that are targeted to the same problem are studied. Finally, a mathematical model is created.

In the second part an implementation of the chosen algorithm is developed. The implementation is performed with C++11 and CUDA.

In the third part a performance of the implementation is tested. Then, the implementation is compared with the existing solutions. Last, a correctness of the implementation is checked against the mathematical model.

After the work was done, the following conclusions were made.

The results show clearly that SIMD computation class devices usage (where CUDA belongs to) is not a sufficient condition to gain performance, even being designed for massive data processing.

The comparison of numerical results with initial measurements of existing rendering approaches confirms that the technique of patch merging increases a number of processed images.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В настоящее время 3D графика находит всё большее применение в различных отраслях человеческой деятельности, начиная с индустрии развлечений и заканчивая промышленностью, медициной и научными исследованиями.

Параллельно происходит развитие различных способов организации архитектуры программного обеспечения. Один из распространённых в настоящее время видов архитектуры, трёхзвенная с «тонким» клиентом, имеет ряд как преимуществ, так и недостатков [11]. К примеру, с одной стороны, уменьшаются требования к устройствам пользователей; а с другой – повышается нагрузка на серверную часть. Как результат, быстродействие серверной части имеет первостепенное значение, особенно при нагрузке, вызванной большим количеством посетителей.

Объединение двух вышеупомянутых направлений образует интернет-ориентированную клиент-серверную систему визуализации 3D графики на стороне сервера. Актуальность работы заключается во всё более возрастающей востребованности подобных систем и их производительности в связи с миниатюризацией пользовательских устройств и всё большем распространении широкополосного доступа в интернет.

Проблема, решаемая в рамках данной работы, заключается в отсутствии публично доступных решений, обеспечивающих возможность одновременной визуализации большого количества (порядка нескольких тысяч) видеорядов с проходом камеры по заданному пользователем пути.

Практическая новизна работы заключается в разработке подхода к оптимизации массовой визуализации видеорядов для одновременного выполнения множества запросов на нагруженной системе.

Объект исследования – процесс построения видеоряда по модели объектов при движении камеры по заданной траектории.

Предмет исследования – параллельные алгоритмы высокопроизводительной визуализации видеорядов на основе заданной траектории движения камеры.

Цель исследования – разработка программного обеспечения для получения наибольшей производительности с точки зрения количества одновременно выполняемых запросов посредством разработки специальных алгоритмов и методов визуализации.

Так как в данной работе рассматривается понятие «видеоряд», рассмотрим его отличие от схожего понятия, «видеопоток»:

1. Видеоряд – это упорядоченная совокупность изображений.

2. «Видеопоток – это видеоряд, подвергнутый сжатию и преобразованию посредством кодека» [19], изначально рассчитанного на работу с видео и недостатками его восприятия зрительной системой человека [29].

Задачами данной работы являются:

1. Оценка современного состояния научного знания по рассматриваемой проблеме.

2. Проведение анализа существующих решений, их достоинств и недостатков.

3. Построение математической модели визуализации видеоряда исходной 3D модели в общем и работы с путями перемещения камеры в частности;

4. Реализация алгоритмов модели формирования видеоряда и его оптимизации.

Практическая применимость данной работы – возможность разработки глобально доступных интерактивных виртуальных миров с широкой областью применения, включая (но не ограничиваясь) детальное визуальное изучение и измерение без необходимости получения твердотельной модели посредством 3D принтера. Подобные миры могут быть использованы, к примеру, в музейном деле в качестве виртуального выставочного зала.

Данная выпускная квалификационная работа состоит из введения, трёх глав и заключения.

В первой главе изучается теоретический материал по выбранной теме и текущий уровень развития отрасли, а также существующие программные средства, направленные на решение проблемы настоящей работы, с последующим формированием математической модели.

Во второй главе осуществляется практическая реализация сформулированной математической модели в виде программы на языке C++11, использующей технологию CUDA.

В третьей главе производится количественная оценка производительности и сравнение её с имеющимися решениями; а также формальное доказательство соответствия программы заявленным требованиям.

Заключение содержит основные выводы по проделанной работе и описывает итоги по выполненным задачам.

1 ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ ЗАДАЧИ НАГРУЖЕННОЙ ВИЗУАЛИЗАЦИИ

1.1 Описание задачи формирования видеоряда

Общая задача заключается в создании компонента, работающего совместно с Web-сервером в тесном взаимодействии с ним. Данный компонент должен принимать от Web-сервера запросы на визуализацию видеоряда с проходом виртуальной камеры по определённой траектории. По результатам работы компонент должен возвращать закодированный видеопоток. В связи с тем, что указанный сервер ориентирован на большое количество (порядка 10^3) одновременно обрабатываемых запросов, то и компонент должен иметь возможность массовой визуализации.

Любая [15] система формирования видеоряда выполняет следующие действий вне зависимости от своего назначения:

1. На вход системы подаются: модель (совокупность многогранников в евклидовом 3D пространстве), текстура (изображение, наносимые на поверхность моделей) и траектория движения камеры (совокупность опорных точек в евклидовом 3D пространстве).

2. Компонент визуализации на основе этих входных данных визуализирует серию статических изображений. Каждое изображение соответствует определённому положению виртуальной камеры на данной траектории.

3. Кодек принимает серию изображений и компонует видеофрагмент, каждый кадр которого соответствует одному входному изображению. Совместно с компоновкой производится сжатие потока согласно используемому формату кодирования. [28]

DFD диаграмма подобной системы представлена на рис. 1.1.

Однако не все компоненты системы представляют интерес в рамках данной работы. Так, модель поступает после подготовки сторонним

инструментарием; кодек же не представляет научной новизны в рамках исследуемой темы, а потому может быть использован уже существующий. В связи с этим в рамках данной работы будут рассмотрены только траектория и компонент визуализации.

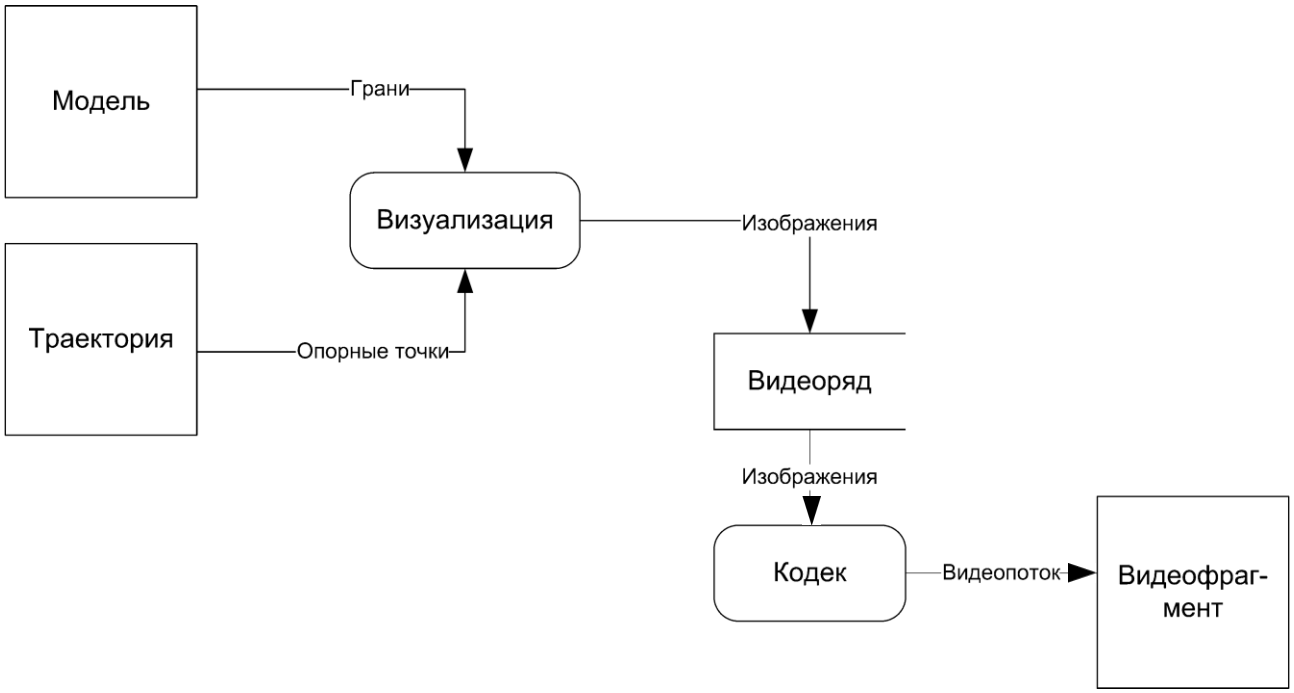

Рисунок 1.1 – DFD верхнего уровня разрабатываемой системы

Перейдём к обзору исследований в смежной области оптимизации нагруженной визуализации. Здесь можно выделить работу Nicolas Tizon, французского сотрудника компании VIDEC, специализирующейся на разработке систем видеотрансляции. Он предлагает [31] оптимизировать передачу видеоряда путём ограничения передачи изменяющимися фрагментами изображения. Однако главной проблемой является не столько малая пропускная способность канала связи с клиентом, сколько неспособность серверного визуализатора одновременно обслужить запросы в большом количестве.

1.2 Обзор существующих решений поставленной задачи

Рассмотрим имеющиеся средства, решающие схожие задачи по исследуемой проблеме.

Существует значительное количество виртуальных туров (в. табл. А.1, (см. Приложение А) приведены наиболее популярные из них), однако технически они представляют собой совокупность заранее сделанных панорамных снимков с возможностью перехода между ними (например, VR-музей ГМИИ им. Пушкина). Также имеются решения, производящие трансляцию в реальном времени средствами стационарно смонтированной камеры на поворотном штативе (предоставляемой в монопольное использование одному из текущих посетителей). В связи с этим можно сделать вывод о том, что полных аналогов разрабатываемому ПО нет.

1.3 Разработка математического обеспечения компонента формирования видеоряда

Выше было сказано, что входными данными модели являются модель (совместно с текстурами) и траектория. Модель и текстуры не обладают научной новизной и не входят в исследуемую проблему, а потому описание исчерпывается общей характеристикой, данной в п. 1.1.

Выходными данными является видеоряд как последовательность изображений в цветовом пространстве RGB. Конкретный формат представления также неуместен в рамках разработки математической модели, так как зависит от конкретной реализации визуализатора.

Вернёмся к рассмотрению входных данных о траектории. Как было сказано выше, она описывается упорядоченной совокупностью контрольных точек, в которых камера обязательно должна быть. Иными словами,

$$P = \{\bar{p} : \bar{p} \in R^3\}, \tag{1.1}$$

$$C \cap P = P \tag{1.2}$$

где I – кортеж входных параметров;

P – данные опорные точки траектории;

$\bar{p}$ – отдельно взятая точка траектории;

C – некая гипотетическая кривая, которую камера фактически описывает во время визуализации видеоряда.

Однако данный набор точек не может быть непосредственно использован. Это связано с тем, что количество кадров (1.3) в видеоряде значительно больше количества опорных точек.

$$N = t_{\max} f \tag{1.3}$$

где N – количество точек на траектории, необходимых для построения видеоряда целиком;

$t_{\max}$ – длина видеоряда, в секундах;

f – кадровая частота видеоряда, кадры/с.

В связи с этим необходим способ получения произвольного количества произвольных точек траектории, если известно положение только нескольких опорных точек – интерполяция [6]. Существует несколько видов интерполяции, каждый из которых обладает своими характеристиками. Однако для всех методов справедливо, что для построения интерполяционного полигона не требуется знать положение всех точек.

Стоит отметить существование явления появления ложных экстремумов, когда экстремум появляется не в контрольной точке, а внутри сегмента [7]. Данное явление следует всеми способами избегать, т. к. оно усложняет исходное построение маршрутов. Пример подобного явления приведён на рис. 1.2, где серым цветом обозначен результат линейной интерполяции, а чёрным – кубической. Сравнительная характеристика различных интерполяционных многочленов приведена в табл. Б.1 (см. Приложение Б). Важно отметить, что многочлены Ньютона, Лагранжа и Эрмита не принимали

участия в рассмотрении в связи с задействованием всех опорных точек, что негативно сказывается на сложности формулы и, как результат, малой скоростью работы её реализации. В случае же со сплайнами Безье часть опорных точек не лежит на образуемой траектории [2].

По результатам исследования был сделан вывод о том, что кубический многочлен (рис. 1.3) является наиболее оптимальным среди рассмотренных интерполирующих методов. По сравнению с линейной интерполяцией (рис. 1.4) он обладает гладкостью первой производной, отсутствие которой привело бы к резкому, скачкообразному изменению угла поворота камеры в окрестности контрольных точек. Косинусная же интерполяция (рис. 1.5) обладает нулевой производной в окрестности контрольных точек, что приводит к прерывистому характеру движения камеры.

Рисунок 1.2 – Демонстрация выброса при интерполяции

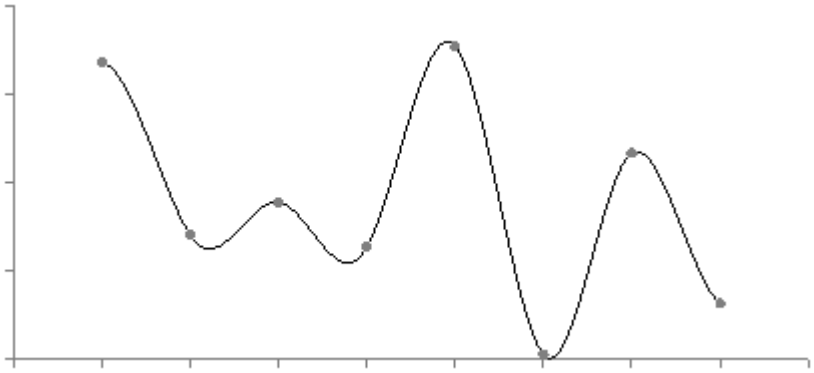

Рисунок 1.3 – Пример кубической интерполяции

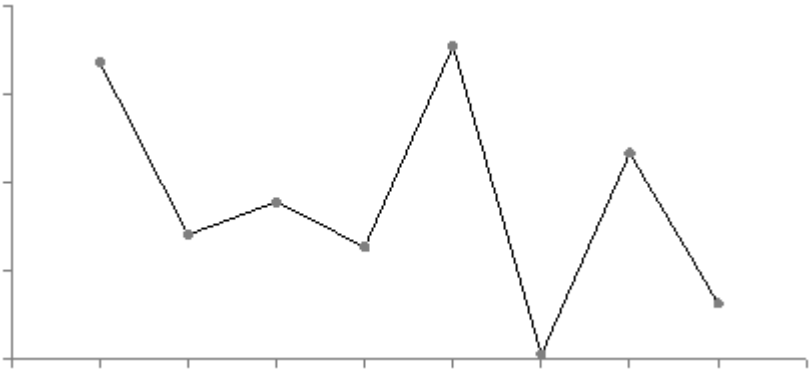

Рисунок 1.4 – Пример линейной интерполяции

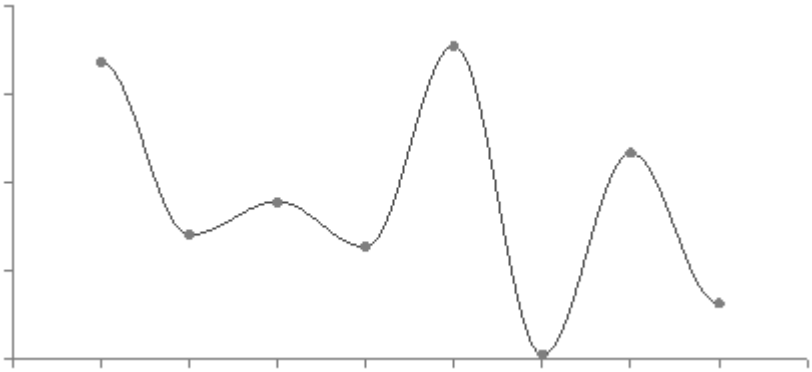

Рисунок 1.5 – Пример косинусной интерполяции

Формула кубического многочлена имеет следующий вид (1.4–1.5) [14].

$$S(t) = at^3 + bt^2 + ct + d, \tag{1.4}$$

$$\begin{cases} a_i = y_i \\ \delta_{i-1} = \dfrac{-h_i}{2h_{i-1} + 2h_i + h_{i-1}\delta_{i-2}}, \delta_1 = \dfrac{,-h_2}{2(h_1 + h_2)} \\ \lambda_{i-1} = \dfrac{3h_i - 3h_{i-1} - h_{i-1}\lambda_{i-2}}{4h_{i-1}}, \lambda_1 = \dfrac{3(h_2 - h_1)}{2(h_1 + h_2)} \\ c_{i-1} = \delta_{i-1}c_i + \lambda_{i-1} \\ d_i = \dfrac{c_i - c_{i-1}}{3h_i} \end{cases} \quad (1.5)$$

где y_i – значение y, соответствующее i-й контрольной точке;

h_i – длина i-го сегмента вдоль оси x;

δ, λ – промежуточные, прогоночные коэффициенты, введённые для сокращения записи.

Несмотря на внешнюю сложность, вычисленные коэффициенты a, b, c и d неизменны для всех точек внутри сегмента. В придачу, эти коэффициенты константны, а потому не требуют пересчёта при взятии производной интерполяционного многочлена.

После выбора интерполирующей кривой необходимо определить, какие именно её точки нас интересуют и как их получить. Пусть вся траектория задана таким образом, что её начальная точка соответствует $t_{\text{общ}} = 0$, а конечная – $t_{\text{общ}} = t_{\max}$, т. е. общей длительности видеоряда. Тогда для получения N кадров (см. формулу (1.3)) необходимо взять N точек траектории с интервалом $\frac{t_{\max}}{N} = \frac{1}{f}$, где f – кадровая частота видеоряда, кадры/с.

Каждая из этих точек, в свою очередь, должна быть отображена на соответствующий сегмент с пересчётом параметра t. Так как разность между позицией начала всей интерполяционной кривой и интересующего нас сегмента является накопительной суммой длин всех предыдущих отрезков [24], рационально будет проводить последовательный расчет координат для всех точек сразу.

На основании вышесказанного была составлена блок-схема (см. рис. В.1 в Приложении В).

При разработке алгоритма возникла необходимость в функции «SegLen», возвращающей длину заданного сегмента. Длина сегмента вычисляется с помощью формулы криволинейного интеграла (1.6):

$$SegLen(S) = \int_a^b \sqrt{x'(t)^2 + y'(t)^2 + z'(t)^2}\,dt \tag{1.6}$$

Подставив в (1.6) формулу (1.4) получаем:

$$SegLen(S_i) = \int_0^{x_{i+1}-x_i} \sqrt{1 + (a_i t^2 + 2b_i + c_i)^2}\,dt \tag{1.7}$$

где S_i – i-й сегмент;

a_i, b_i, c_i – коэффициенты (1.5), вычисленные для i-го сегмента.

Теперь переходим к непосредственно применению интерполированной кривой. Она требуется для определения положения виртуальной камеры, проецирующей входную модель на плоскость экрана.

Рассмотрим сам камеру подробнее. Управление её характеристиками (такими, как положение, поворот, угол обзора, пропорции итогового изображения) осуществляется матрицей 4x4. Данная матрица используется следующим образом: к каждой вершине каждой модели применяется цепочка преобразований [8] вида:

$$\bar{q} = \left(\frac{u.x}{u.w}, \frac{u.y}{u.w}, \frac{u.z}{u.w}\right), \quad \bar{u} = AB\bar{p} \tag{1.8}$$

где $\bar{q}$ – итоговые, спроецированные координаты точки, полученные проективным делением проективного пространства;

$\bar{u}$ – точка в проективном пространстве;

A – матрица линейных преобразований камеры;

B – матрица прочих, предшествующих линейных преобразований;

$\bar{p}$ – исходный вектор в виртуальном R^3 (т. е. 3D) пространстве, вида $\bar{p} = (x, y, z, 1)^T$. Четвёртая размерность, изначально равная единице была введена с целью расширения 3D пространства до проективного.

Матрицу камеры A из формулы (1.8) «можно разбить на произведение двух других матриц: линейной в R^3 и линейной в проективной R^4» [25]. В этом случае первая описывает перемещение и поворот, а вторая – проекцию. Так как матрица проецирования линейно независима [10] от матриц перемещения и вращения, то в данной работе она рассматриваться не будет. Остановимся подробнее на матрице линейных (на R^3) преобразований. Она имеет вид

$$T = \begin{pmatrix} x_x & x_y & x_z & \Delta x \\ y_x & y_y & y_z & \Delta y \\ z_x & z_y & z_z & \Delta z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{1.9}$$

где T – матрица линейных преобразований, описанная выше;

x_i, y_i, z_i – пропорциональные передаточные коэффициенты, передают вращение;

Δx, Δy, Δz – константные передаточные коэффициенты, передают перемещение.

Приступим к определению значения коэффициентов. Матрица (1.9) задаёт поворот и перемещение в виде перехода к новому реперу, координаты базисных векторов которого указаны в строках матрицы относительно текущей системы координат. В связи с тем, что мы выполняем переход из текущей системы координат в новую, необходимо выполнить транспонирование T.

Так как матрица (1.9) задаёт поворот в виде перехода к новому базису, нам необходимо его выразить. На рис. 1.6 наглядно продемонстрирован целевой базис, состоящий из касательной, нормали и бинормали.

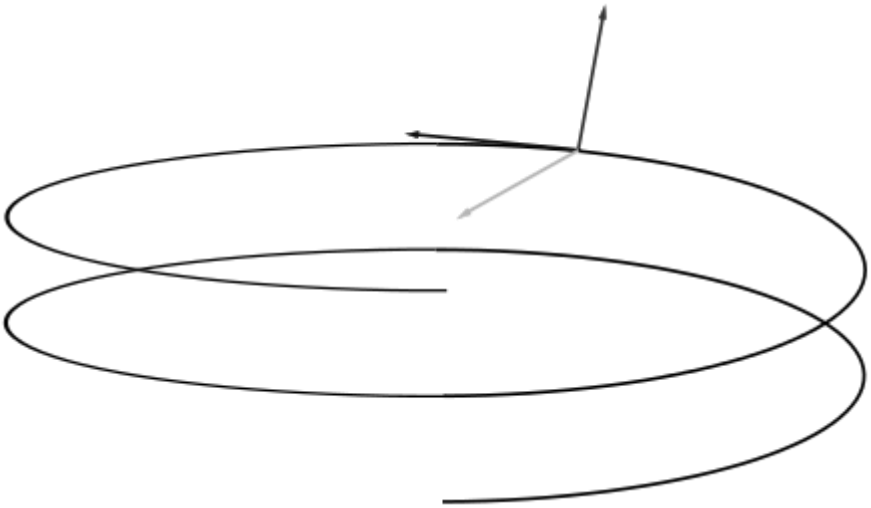

Рисунок 1.6 – Касательная, нормаль и бинормаль к кривой

Наиболее тривиальной для выражения является бинормаль. Пусть камера не совершает вращательных движений в плоскости, параллельной плоскости проекции. Тогда бинормаль всегда будет направлена вверх. Учитывая, что длину бинормали может изменить только поворот в вертикальной плоскости, имеет смысл нормировать бинормаль на угол поворота для сохранения его единичной длины:

$$\bar{b} = \bar{k} \sin \alpha \tag{1.10}$$

где $\bar{b}$ – нормированная бинормаль;

$\bar{k}$ – исходная бинормаль;

α – угол наклона камеры в вертикальной плоскости, перпендикулярной плоскости проекции.

Направление касательной определяется значением градиента (1.11) интерполяционной функции (1.4):

$$\begin{cases} T_x = \dfrac{\partial S_x}{\partial t} = 3a_x t^2 + 2b_x + c_x \\ T_y = \dfrac{\partial S_y}{\partial t} = 3a_y t^2 + 2b_y + c_y \\ T_z = \dfrac{\partial S_z}{\partial t} = 3a_z t^2 + 2b_z + c_z \end{cases} \quad (1.11)$$

где T_i – значение проекции вектора касательной на ось i;

a_i, b_i, c_i – коэффициенты (1.5), вычисленные для i-го сегмента.

Наконец, нормаль выражается как скалярное произведение двух других базисных векторов ортонормированного базиса.

После нахождения всех составляющих новой системы координат, их необходимо нормировать во избежание линейных искажений при переходе между системами координат.

Перейдём к рассмотрению оптимизаций, в частности оптимизации траектории. Она обусловлена тем, что в силу ограниченности пространства виртуального мира траектории с большой вероятностью могут проходить достаточно близко друг к другу или пересекаться. При этом увеличение количества траекторий (то есть количества запросов) также увеличивает и вероятность полного или частичного их совпадения.

Примем, что фрагменты траектории являются совпадающими, если расстояние между двумя соответствующими кривыми не превышает заданного порога ε.

Формализуем всё вышесказанное. Фрагменты траекторий накладываются, если:

$$\forall t_{11}, t_{12} \in [0;1], t_{21} \in [0;1], t_{12} \in [0;1] \quad [t_{11}; t_{21}] \subset S_1,\ [t_{12}; t_{22}] \subset S_2, O(S_1, S_2) < \varepsilon, \quad (1.12)$$

$$O(S_1, S_2) = \int_a^b \sqrt{S_1(t)^2 + S_2(t)^2}\, dt \quad (1.13)$$

где t_{11} – временной параметр начала отрезка P_1;

t_{21} – временной параметр конца отрезка P_1;

t_{12} – временной параметр начала отрезка P_2;

t_{22} – временной параметр конца отрезка P_2;

$O(S_1, S_2)$ – расстояние между S_1 и S_2 по метрике L_1 (т. е. евклидовой);

ε – порог различения.

Порог различения был введён по следующим причинам:

1. Конечная точность машинного представления чисел с плавающей запятой и, как следствие, конечность машинного ε.

2. Низкая точность пользовательского ввода.

3. Малая степень отличия видеорядов в случае перемещения камеры по двум практически идентичным траекториям.

4. Пространственная дискретность видеоряда, из которой следует дискретность положений и направления взгляда (направляющей) виртуальной камеры, влияющих на итоговое изображение

5. Возможность оптимизации собственно траектории путём её упрощения. Упрощение достигается уменьшением количества контрольных точек при объединении ранее визуализированных более простых траекторий

Каждой траектории сопоставлен свой видеоряд в отношении «один к одному» Сопоставление выполнено следующим образом (1.14):

$$\{S, t\} \to F, \quad S \in T, t \in [0,1] \cap \left\{ \left| \frac{\partial S}{\partial t} \right| \cdot \frac{i}{f}, i \in N_0 \right\}, F \in \{R^2 \to R^3\} \tag{1.14}$$

где S – кривая, описывающая маршрут;

t – параметр, однозначно определяющий точку на S;

T – совокупность всех маршрутов;

f – кадровая частота видеоряда, являющегося полезной нагрузкой сопоставления;

$\left|\frac{\partial S}{\partial t}\right|$ – функция, описывающая скорость движения камеры вдоль S как зависимость скорости (производной первого порядка) от положения на кривой.

Иными словами, каждой паре «маршрут – дискретное положение» сопоставлен кадр видеоряда.

В результате всего вышеописанного можно сделать следующие выводы:

1. Ключевым элементом определения траектории движения камеры является интерполяция. В зависимости от вида выбранного интерполяционного многочлена траектория способна претерпеть значительные изменения.

2. Для возможности отображения положения на траектории в параметры виртуальной камеры было необходимо ввести ряд ограничений, включая запрет на поворот камеры в плоскости, параллельной плоскости проекции, и ограничение на допустимое направление ориентации камеры.

3. Наиболее эффективный способ оптимизации скорости визуализации – полное устранение визуализации и повторное использование ранее созданных фрагментов видеорядов.

2 ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ РАЗРАБОТАННОЙ МОДЕЛИ

2.1 Краткая характеристика целевой вычислительной системы

Nvidia CUDA – это архитектура распределённых вычислений, ориентированная на выполнение алгоритмов общего назначения на GPGPU (т. н. видеокартах). [16]

1. Тип архитектуры – SIMD (Single Instruction Multiple Data, одна инструкция – множество данных); процессорные ядра очень просты и многочисленны (около 10^3) [30]. Вычисления организованы в виде масштабируемой иерархии «грид – блок – поток». Из этого следует, что реализация алгоритма должна представлять собой массовую и однотипную обработку большого массива данных.

При реализации алгоритма на CUDA требуется учитывать следующие ограничения [5]:

1. Отсутствие поддержки рекурсивных вызовов в силу линейности процесса исполнения кода на GPU. В свою очередь, строгая линейность (т.е. отсутствие ветвлений) обусловлена необходимостью выполнения одного и того же кода для абсолютно всех обрабатываемых данных вне зависимости от значения этих данных.

2. Ограниченная поддержка циклов и ветвлений. Во избежание потери производительности необходимо произвести ряд замен на всех алгоритмах, исполняемых на видеокарте:

2.1. Использование циклов только с фиксированным количеством итераций;

2.2. Замена условных ветвлений на явное вычисление значений для обеих ветвей цикла с последующим выбором одной из двух переменных в итоговую. Подобное действие должно быть не условным, а линейным, поэтому уместно использовать интерполирующую функцию с условием в качестве коэффициента интерполяции.

2.3. Наличие раздельной памяти для CPU и GPU. Относительно небольшая скорость обмена данными между устройствами [21] обязывает производить все действия согласно конвейерному принципу. Данный принцип предполагает начальную загрузку исходных данных в память GPU (DRAM), последовательный вызов программ на стороне GPU для обработки этого буфера и итоговую его выгрузку обратно в основное ОЗУ (RAM).

3. Взаимоблокировки при одновременном обращении различных потоков в один и тот же регион памяти. Это вынуждает тщательно планировать распределение данных по потокам, а также избегать взаимодействия потоков между собой.

4. Однотипность выполняемой программы для всего потока данных. Причиной этого является суперскалярность [4] – малое количество вычислительных устройств при большом количестве регистров.

5. Нет возможности предсказать очерёдность запуска блоков в гриде. Как результат, отсутствует возможность детерминировано предсказать готовность очередного блока данных для его непосредственного получения. Из-за этого необходимо ожидание окончания последовательного исполнения всех блоков при нехватке ресурсов видеокарты для их параллельного выполнения.

Общий подход к программированию на SIMD-платформах [9]:

1. Разбить задачу на элементарные блоки данных, над которыми выполняется стандартный алгоритм обработки, единый для всех блоков.

2. Разбить загружаемые и выгружаемые данные на элементарные непересекающиеся блоки, которые каждый из процессов прочтёт или запишет.

3. Определить конфигурацию грида/блока, позволяющую оптимальное размещение промежуточных данных в регистрах и общей памяти.

4. Определить оптимальную вычислительную загрузку потоковых процессоров.

5. Определить график совместного обращения к памяти процессами при загрузке данных.

2.2 Программная реализация алгоритма

2.2.1 Формулирование требований

Первичной частью проектирования и разработки программного обеспечения является формулирование требований, функциональных и нефункциональных. Формулирование требований в рамках данной работы было выполнено согласно нотации FURPS+.

FURPS+ обозначает совокупность следующих категорий требований [26]:

1. Functionality – функциональные требования: свойства, возможности, безопасность.

2. Нефункциональные требования:

2.1. Usability – требования к удобству использования: человеческий фактор, эстетика, последовательность, документация.

2.2. Reliability – требования к надежности: частота возможных сбоев, отказоустойчивость, восстанавливаемость, предсказуемость устойчивости.

2.3. Performance – требования к производительности: время отклика, использование ресурсов, эффективность, мощность, масштабируемость.

2.4. Supportability – требования к поддержке: возможность поддержки, ремонтопригодность, гибкость, модифицируемость, модульность, расширяемость, возможность локализации.

2.5. Дополнительные требования.

Сами требования приведены в Приложении Г.

Однако для корректного составления функциональной части требований требуется описание системы с точки зрения вариантов её использования. Данный подход позволяет рассмотреть разрабатываемую систему с точки зрения конечного пользователя, персонала, а также внешних событий, а также связанных с ними действий. Диаграмма вариантов использования, связанная с фундаментальным использованием системы приведена в Приложении Д (табл. Д.1)

2.2.2 Характеристика формата входных и выходных данных

Если в рамках разработки математической модели система рассматривалась обособленно и в отрыве от программно-аппаратного окружения, то при реализации необходимо решить задачу оценки среды выполнения. Результатом данной оценки стала UML диаграмма развёртывания, приведённая в Приложении Е. Вот её кратное описание:

1. Существует три различных устройства:

1.1. Клиентское, на котором развёрнут тонкий клиент, реализованный в виде интернет-обозревателя.

1.2. Web-сервер. Выступает в роли слоя логики, поддерживая диалог с клиентом, собирая информацию для создания очередного запроса на визуализации и принимая обратную связь.

1.3. Сервер визуализации (т. н. «рендер-ферма»). Представляет собой специализированный компьютер, специализированный на CUDA-вычислениях.

1.4. Хранилище. В связи с ограниченными возможностями сервера визуализации по хранению кэшей (обусловлено требованиями к производительности, в которые не укладываются общесистемные задержки на ввод/вывод) вынужденной мерой стало введение дополнительного хранилища за пределами сервера визуализации. Данный перенос основан на том принципе, что «работа с дисковыми накопителями является блокирующей на уровне системы из-за того, что диск неспособен обрабатывать несколько запросов от разных программ одновременно» [20]; в то же время «сетевые задержки являются неблокирующими, поступившие пакеты распределяются по приложениям с большой скоростью» [22].

В связи с тем, что разрабатываемая система визуализации взаимодействует с другими системами, как описано выше, необходимо рассмотреть и форматы для обмена данными.

1. Формат хранения моделей – STL. Является распространённым в сфере 3D печати, сканирования и моделирования [17]. Описывает модель в виде

совокупности независимых треугольных граней, их нормали и позиции вершин. Однако в связи с ориентированностью на 3D печать он не поддерживает наложение текстур на модели.

2. Формат хранения текстур – BMP. Является форматом, хранящим изображения без сжатия, что позволяет получать текстуры максимально возможного качества. Несмотря на полное отсутствие назначения текстур моделям (причина – отсутствие повершинных текстурных координат), было найдено обходное решение, описанное ниже, в п. 2.2.7.

3. Формат выходного видеоряда – упорядоченная последовательность изображений в формате BMP. Необходимость передачи несжатых и необработанных кадров кодеку (рис. 1.1).

4. Формат хранения опорных точек – массив, сериализованный в JSON. Каждый элемент этого массива представляет собой объект с тремя атрибутами, задающими положение точки в пространстве: «x», «y» и «z».

Следующий этап – это переложение разработанной математической модели на целевую платформу. Данное действие связано с тем, что NVidia CUDA – это технология с рядом особенностей и ограничений (как было описано выше), связанных с массовостью вычислений. Так как целью данной работы является достижение наибольшей производительности, эти особенности необходимо учитывать при разработке, что и было сделано далее.

2.2.3 Общая последовательность действий

При приёме запроса на визуализацию выполняется следующий ряд действий:

1. Извлечение опорных точек траектории из запроса.
2. Построение интерполирующего полинома.
3. Загрузка ускоряющей структуры (BSP).

4. Выполнить разбиение полинома таким образом, чтобы каждый фрагмент целиком принадлежал определённому узлу существующей ускоряющей структуры.

5. Для каждого фрагмента полинома осуществить поиск наиболее близко расположенного фрагмента. Проверка найденного фрагмента: если он не удовлетворяет условию, найти следующий по расстоянию фрагмент и проверить его; повторять, пока не будет найден фрагмент, удовлетворяющий условию, либо количество итераций не превысит порог.

6. Убрать из исходной структуры те фрагменты, для которых было найдено соответствие в ускоряющей структуре. Оставить ссылку на месте убранного фрагмента.

7. Выполнить последовательный перебор всех фрагментов оптимизированной траектории из запроса. Для каждого фрагмента:

7.1. Если фрагмент был замещён: извлечь из ускоряющей структуры полезную нагрузку (фрагмент видеоряда), соответствующий замещаемому узлу и передать в качестве выходных данных.

7.2. Если фрагмент не был замещён:

7.2.1. Запустить отложенное перестроение ускоряющей структуры с учётом наличия нового фрагмента. Получить временный описатель для размещения полезной нагрузки во время перестроения.

7.2.2. По ходу визуализации фрагмента передавать его в качестве выходных данных, и в качестве полезной нагрузки ускоряющей структуры.

7.2.3. При нехватке дискового пространства для записи очередного кадра фрагмента видеоряда удалить самый редко используемый фрагмент видеоряда и соответствующий ему фрагмент траектории, затем перезапустить отложенное перестроение (в среднем на производительность не влияет, т. к. является редко возникающим и потому исключительным случаем). Важное замечание: данная стратегия вытеснения означает, что один и тот же маршрут, визуализация которого была запрошена сначала до, а потом после вытеснения, может

различаться. Однако данную изменчивость можно рассматривать как возможность достижения оптимального разбиения при поступлении большого количества запросов, либо изменении характера построения траекторий [27].

7.2.4. Перед переходом к следующему фрагменту уведомить хранилище о необходимости применения перестроенной структуры (т. е. персистенции).

Всё вышеперечисленное можно формализовать в виде диаграммы последовательности, приведённой на рис. 2.1.

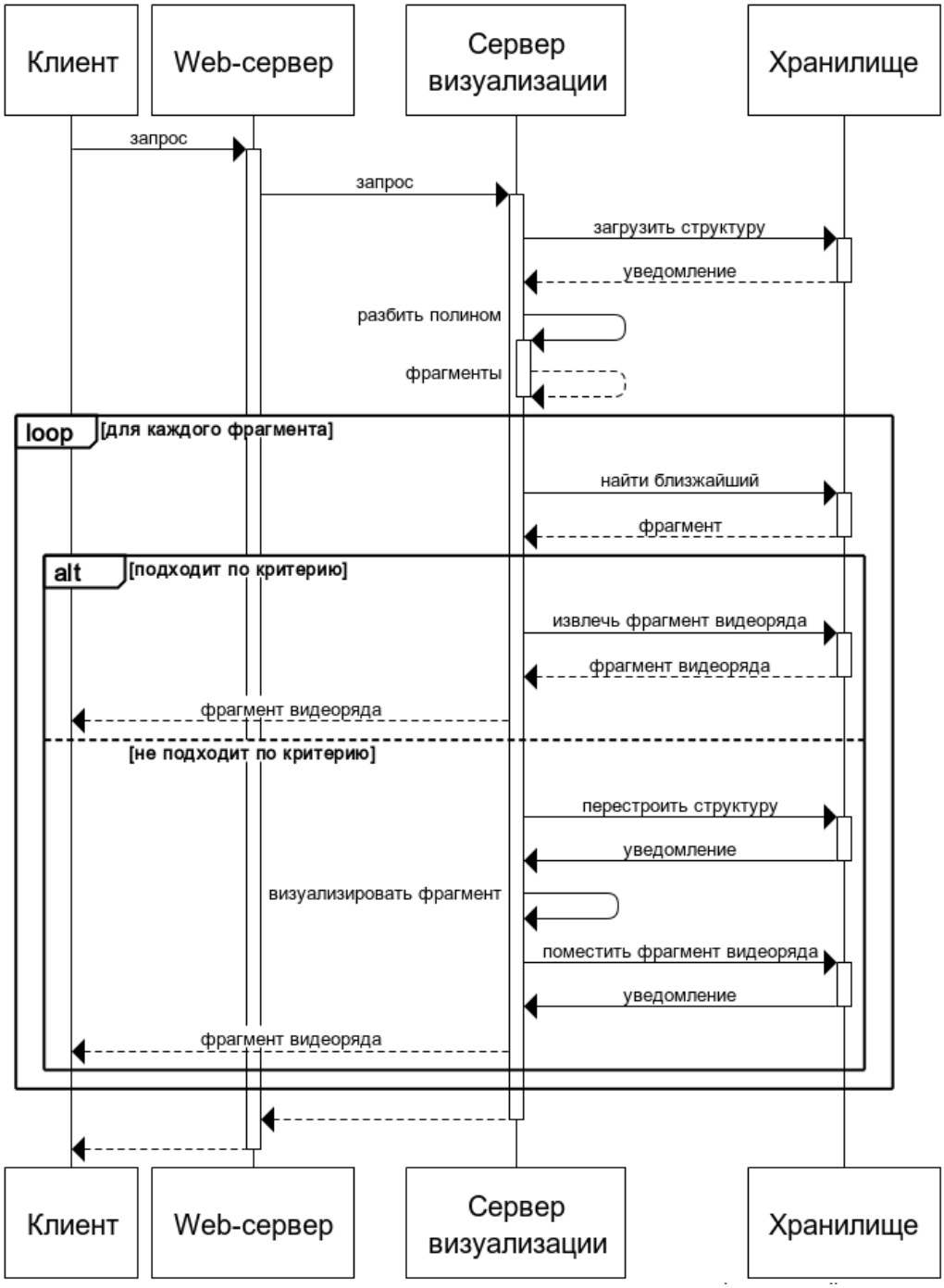

Рисунок 2.1 – диаграмма последовательности разработанной системы

Рассмотрим каждый из вышеперечисленных шагов подробнее.

2.2.4 Извлечение опорных точек из запроса и построение интерполирующего полинома

На данном этапе производится разбор данных в формате JSON. Объекты и атрибуты, не предусмотренные описанием выше, игнорируются.

Во время процесса разбора производится расчет коэффициентов интерполирующего многочлена по формуле (1.5) для каждого из сегментов, ограниченных парой смежных точек. Соответствующий алгоритм приведён на рис. Ж.1.

2.2.5 Загрузка ускоряющей структуры

На следующем этапе производится подготовка и загрузка ускоряющей структуры (если этого не было сделано ранее). Цель её применения – уменьшение времени поиска участка траектории, ближайшего к заданной точке. Это достигается уменьшением асимптотатической сложности с квадратичной ($O(n^2)$ [13]) до сверхлинейной, линейной, логарифмической или даже константной.

В качестве ускоряющей структуры была выбрана BSP – двоичное разбиение пространства (асимптотатическая сложность – $O(\log n)$ [23]); а именно такой её вариант, который хранит объекты в узлах (см. рис. 2.2).

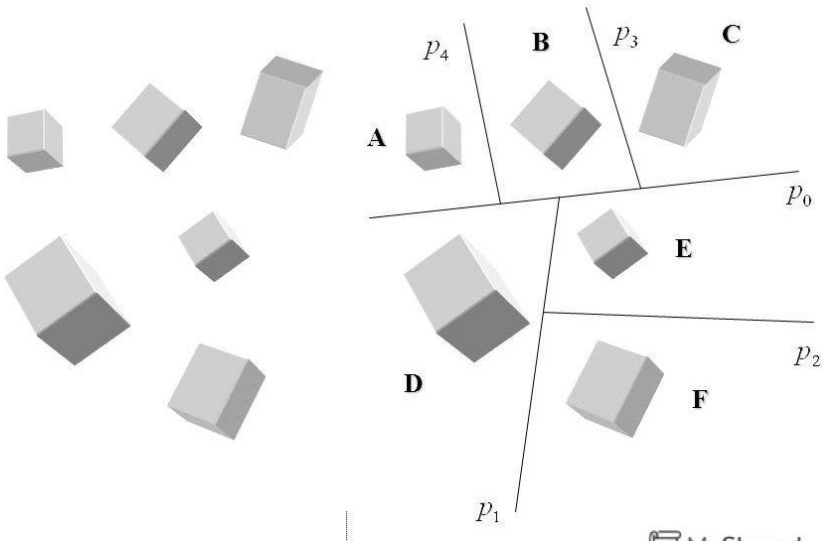

Рисунок 2.2 – Принцип организации данных в используемом варианте BSP

Выбор BSP предоставляет также возможность побочную возможность разбиения протяжённых объектов (в данном случае интерполирующих кривых) на короткие сегменты (см. рис. 2.3, ряды чисел обозначают путь от вершины

двоичного дерева до листа, соответствующего подписанной области). Причём медианная длина сегментов обратно пропорциональна количеству и длине кривых. В связи с этим появляется возможность не уточнять границы условно совпадающего участка, а повторно использовать либо отвергнуть участок целиком

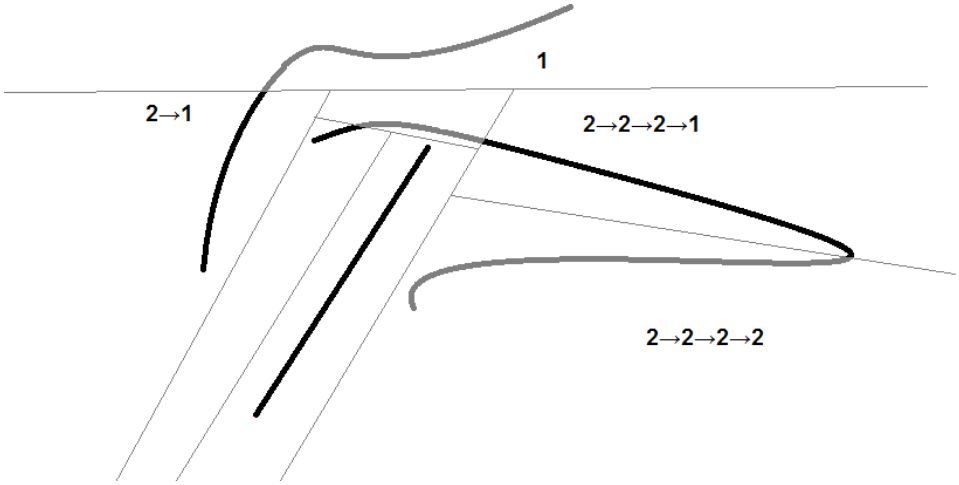

Рисунок 2.3 – BSP-разбиение для протяжённых объектов

На ускоряющую структуру, используемую в данной работе, накладывается два ограничения с целью более достоверного определения ближайших фрагментов: по результатам разбиения каждый узел и соответствующая часть пространства должны содержать объекты со следующими свойствами:

1. Отсутствие разрывов кривой, т. е. «смежность всех сегментов участка, принадлежащего одному и тому же узлу» [1].

2. Ненулевая вторая производная участка, т. е. фрагмент кривой не должен совершать разворотов.

Однако стоит заметить, что данная структура не является самоцелью существования хранилища. Она является средством быстрого доступа к фрагментам видеоряда, назначенным соответствующим узлам дерева. Иными словами, данную структуру можно рассматривать как ассоциативный массив, ключом которого являются полученные в результате разбиения фрагменты

путей, а полезной нагрузкой – видеофрагменты. При этом состав каждого видеофрагмента изменяется при перестроении дерева в результате переноса начальных и конечных частей между фрагментами.

Ускоряющая структура поддерживает следующие запросы:

1. Выборка кривой, являющейся k-й в порядке удаления от заданной точки. Данная операция разбивается на две: определение стратегии рекурсивного выбора очередного узла и определение расстояния от точки до фрагмента кривой. Стратегия выбора заключается в переходе к узлу дерева, содержащему листья, смежные с листом с данной точкой. Смежность узла, в свою очередь, определяется смежностью полупространства всех узлов в цепочке от корня до этого узла. Расстояние же от точки до кривой определяется как длина кратчайшего перпендикуляра к сегменту, проходящему через данную точку, и выражается как экстремум функции (2.1) при условии (2.2):

$$d(x, y, z) = \sqrt{(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2} \to \min, \tag{2.1}$$

$$\begin{cases} x(t) = a_x t^3 + b_x t^2 + c_x t + d_x, \\ y(t) = a_y t^3 + b_y t^2 + c_y t + d_y, \\ z(t) = a_z t^3 + b_z t^2 + c_z t + d_z \end{cases} \tag{2.2}$$

где x_0, y_0, z_0 –точка, для которой производится поиск ближайшей кривой;

t – параметр, однозначно определяющий точку на кривой S.

Поиск экстремума производится для расстояния от искомой точки до точки кривой. Ограничением же является само уравнение кривой (1.4).

Стоит отметить, что строго иерархическая структура данных позволяет не загружать дерево в память целиком, а ограничиваться фрагментами, требуемыми в рамках данной задачи.

2.2.6 Реализация построения интерполированной траектории

Как было указано в математической модели, интерполирование производится посредством интерполяционных полиномов. Цель

интерполирования – получение совокупности пар «момент времени – положение камеры». Можно заметить, что каждый кадр визуализируется независимо; следовательно, и расчёт положения точек на траектории также допустимо выполнять с параллелизмом на уровне данных, распределяя один момент времени t на поток.

Рассмотрим конфигурацию, состоящую из трёхмерного грида, 64 потока на блок. Количество блоков по одной размерности, которое при этом потребуется, вычисляется по формуле:

$$p = \left\lceil \sqrt[3]{\frac{d}{64}} \right\rceil \tag{2.1}$$

где p – размерность грида по определённой координатной оси;

d –количество точек, для которых необходимо рассчитать положение.

Сам алгоритм расчёта не претерпел каких-либо изменений, связанных с переводом из однопоточной реализации в многопоточную (см. рис. 2.4). Переменные, используемые в алгоритме: TLen – накопленная длина сегментов, Tmax – общая длина траектории, loc – массив позиций визуализации.

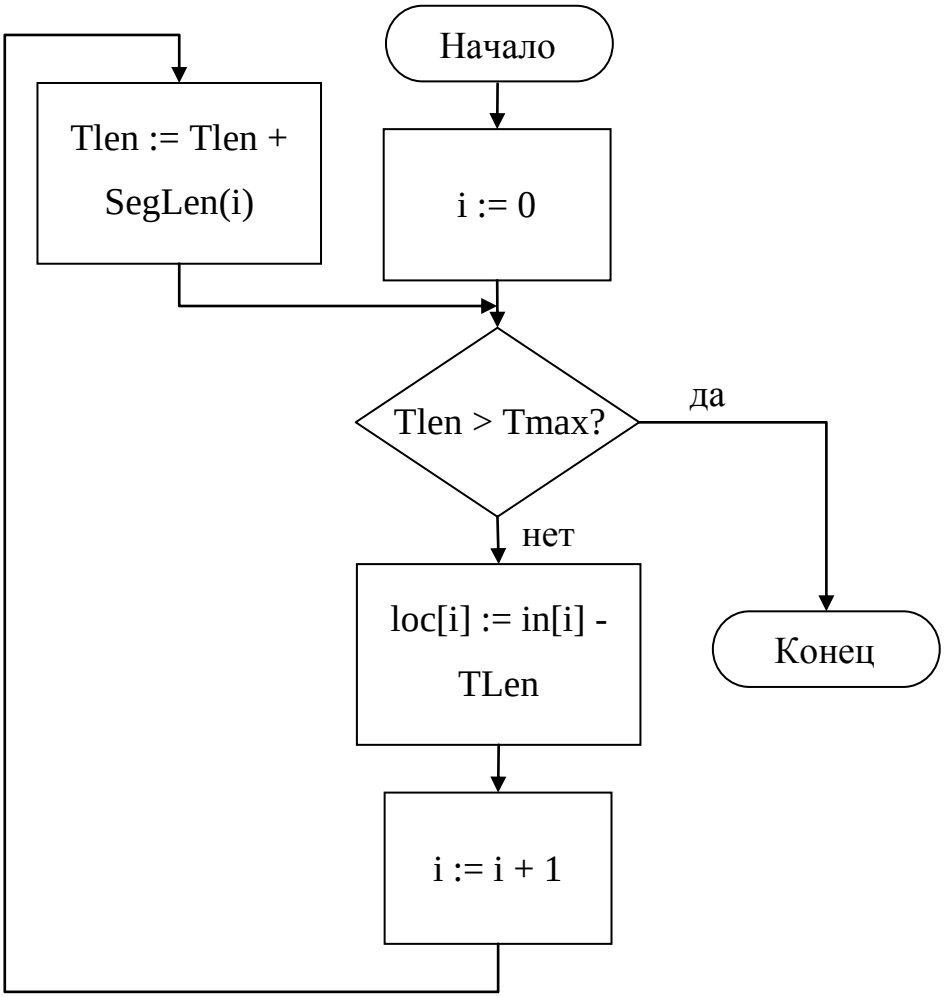

Рисунок 2.4 – Блок-схема алгоритма получения точек траектории, в которых необходимо провести визуализацию

2.2.7 Реализация объединения траекторий

Алгоритм объединения траекторий является наиболее важным кандидатом на распараллеливание, так как его изначальная суть заключается в попарном сравнении всех кривых (см. рис. 2.5), что может быть выполнено независимо и параллельно для каждой пары.

Однако данный подход не является оптимальным, т. к. из $n(n-1)$ попарных сравнений успешным будет только одно. В связи с этим требуется выработка алгоритма, способного снизить совокупное количество сравнений.

Для оптимизации данного алгоритма необходимо применение следующих принципов:

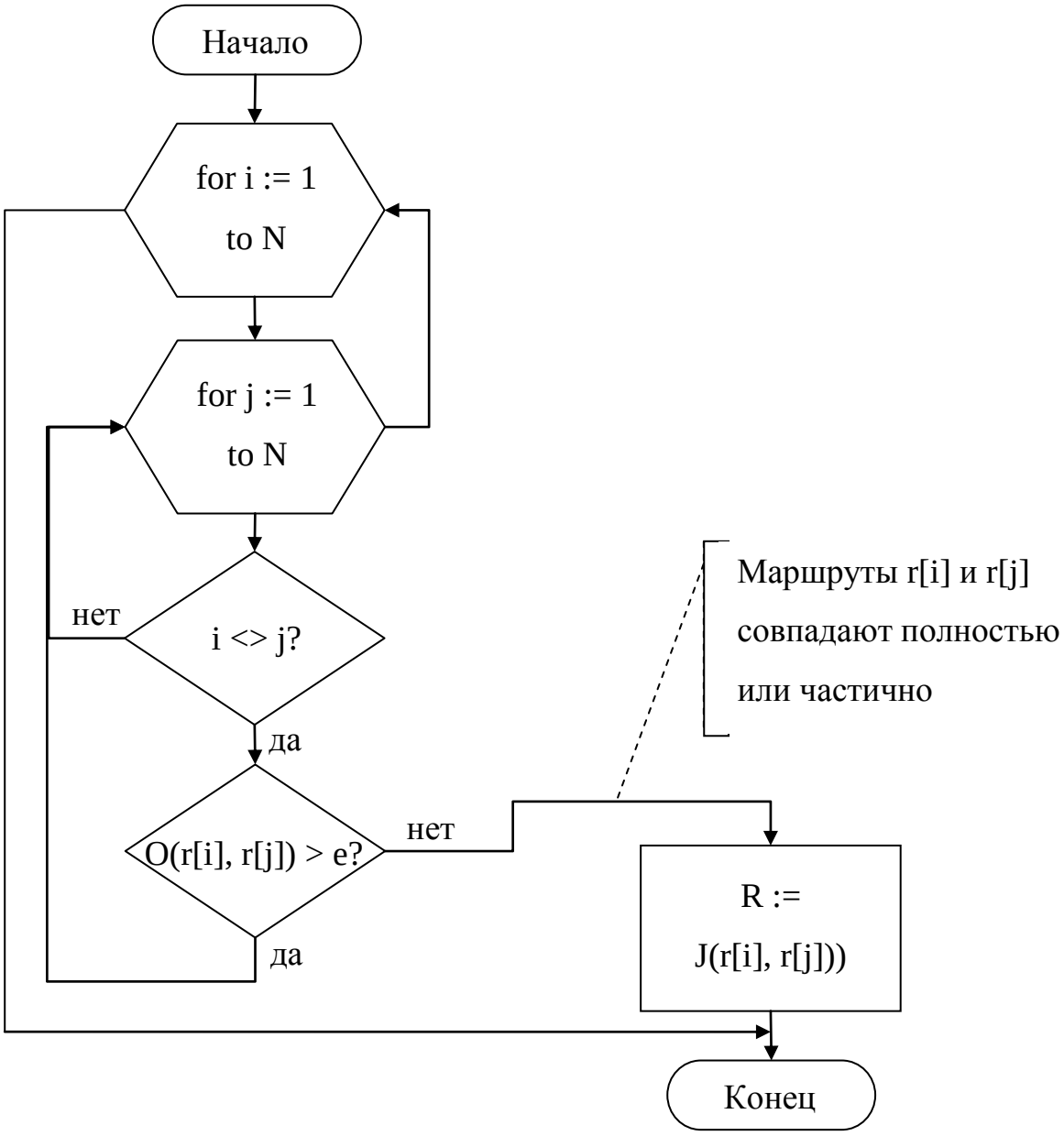

Рисунок 2.5 – Блок-схема алгоритма поиска совпадающих траекторий

1. Разбиение траектории на большое количество фрагментов, что позволяет избавиться от необходимости уточнения и пересчёта границ применяемой кривой в случае её неполного совпадения.

2. Обеспечение непрерывности производной на стыке исходного и замещённого фрагментов.

3. Пересчёт дерева и повторное разбиение фрагментов при добавление новых фрагментов в ускоряющую структуру.

Поиск участка фрагмента для замещения производится следующим образом:

1. Вводится т. н. «текущая точка», относительно которой и ведётся поиск. Эта точка помещается в начало траектории.

2. К хранилищу выполняется запрос на поиск первого ближайшего к этой точке фрагмента. Если ничего найдено не было, помещаем всю траекторию в хранилище.

3. Если расстояние от найденного фрагмента до траектории не превышает заданного порога ε, выполняем подстановку найденного фрагмента и переходим к шагу 6.

4. К хранилищу выполняется запрос на поиск второго ближайшего к этой точке фрагмента.

5. Если расстояние от найденного фрагмента до траектории не превышает заданного порога ε, выполняем подстановку найденного фрагмента траекторию в хранилище.

6. Помещаем в хранилище участок траектории от текущей точки до начала найденного фрагмента.

7. Перемещаем конечную точку визуализируемого участка для совпадения с начальной точкой извлечённого фрагмента. Если конечная точка не лежит на конце траектории – перейти к п. 1.

2.2.8 Реализация применения текстур к STL-моделям

Как было сказано в п. 2.2.1, формат STL не поддерживает хранение текстур и повершинных текстурных координат. Однако спецификация предусматривает [3] обязательное наличие у каждого многогранника двухбайтового поля атрибутов, а также 80-байтный заголовок неспецифицированного содержания в начале файла. Его использование не регламентируется стандартом и остаётся на усмотрение разработчика конкретной программы, работающей с данными типами файлов.

Соответственно, мы можем использовать данное поле для хранения собственных данных. К этим данным относятся: идентификатор текстуры на весь треугольник и текстурные координаты на плоскости треугольника.

Однако эти данные не помещаются в двухбайтовое поле атрибутов. В связи с этим имеет смысл размещение вспомогательных данных в конце STL-файлов после окончания данных, описанных спецификацией (она предписывать явно хранить количество треугольников, и не специфицирует содержимое файла после описания последней грани).

В связи с этим было принято решение вносить в каждый обрабатываемый файл следующий список модификаций:

1. Указывать в начальной заголовке файла сигнатуру – специальную последовательность байтов, позволяющую разрабатываемому компоненту визуализации определять наличие описываемых дополнений. В рамках данной работы была выбрана последовательность «prs.yarygin.hload_renderer.stl_extra», в кодировке «ASCII» завершаемая нуль-символом (символом с кодом ноль, обозначающим конец строки) и четырёхбайтовым числом с количеством элементов вспомогательной таблицы. Данная последовательность уникальна (так как определяет пространство имён по автору и названию приложения), и её длина составляет 42 байта, что меньше верхнего предела в 80 символов.

2. Хранить следом за списком граней списка дополнительных атрибутов (см. рис. 2.1) количество записей с дополнительными атрибутами.

На основе вышеперечисленных пунктов была составлена диаграмма классов, представленная на рис. 2.6, и блок-схема (см. рис. 2.7).

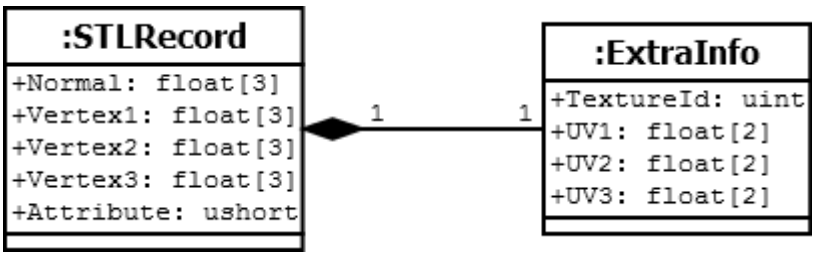

Рисунок 2.6 – Диаграмма классов, описывающая отношения между данными в STL файле и дополнительными данными

Можно заметить, что представленный метод записи дополнительных атрибутов обладает обратной совместимостью с STL-файлами, не обработанные с целью добавления информации и текстурах и их размещении.

Перейдём к обработке ошибок во входном файле. К ним можно отнести:

1. Несуществующий путь к текстуре (текстурам);

2. Некорректный или неподдерживаемый формат текстуры;

3. Идентификатор текстуры, превышающий количество текстур, упомянутых в STL-файле.

Данные ошибки необходимо рассматривать как невосстановимые и прекращать обработку запроса при встрече подобных ошибок с последующей записью в журнал.

Альтернативный вариант игнорирования подобных ошибок с отображением модели без текстуры не был принят в связи с двумя факторами:

1. Модель и связанные с ней текстуры подготавливаются персоналом, а затем многократно используются пользователями. Таким образом, снижается вероятность допущения ошибки. В то же время сам факт допущения следует рассматривать как признак наличия более серьёзной ошибки, требующей внимания персонала.

2. Неполнота визуального восприятия. Отсутствие текстур с замощением всей поверхности модели текстурой-заменителем неприемлемо из соображений нарушения взаимодействия пользователей и виртуального мира.

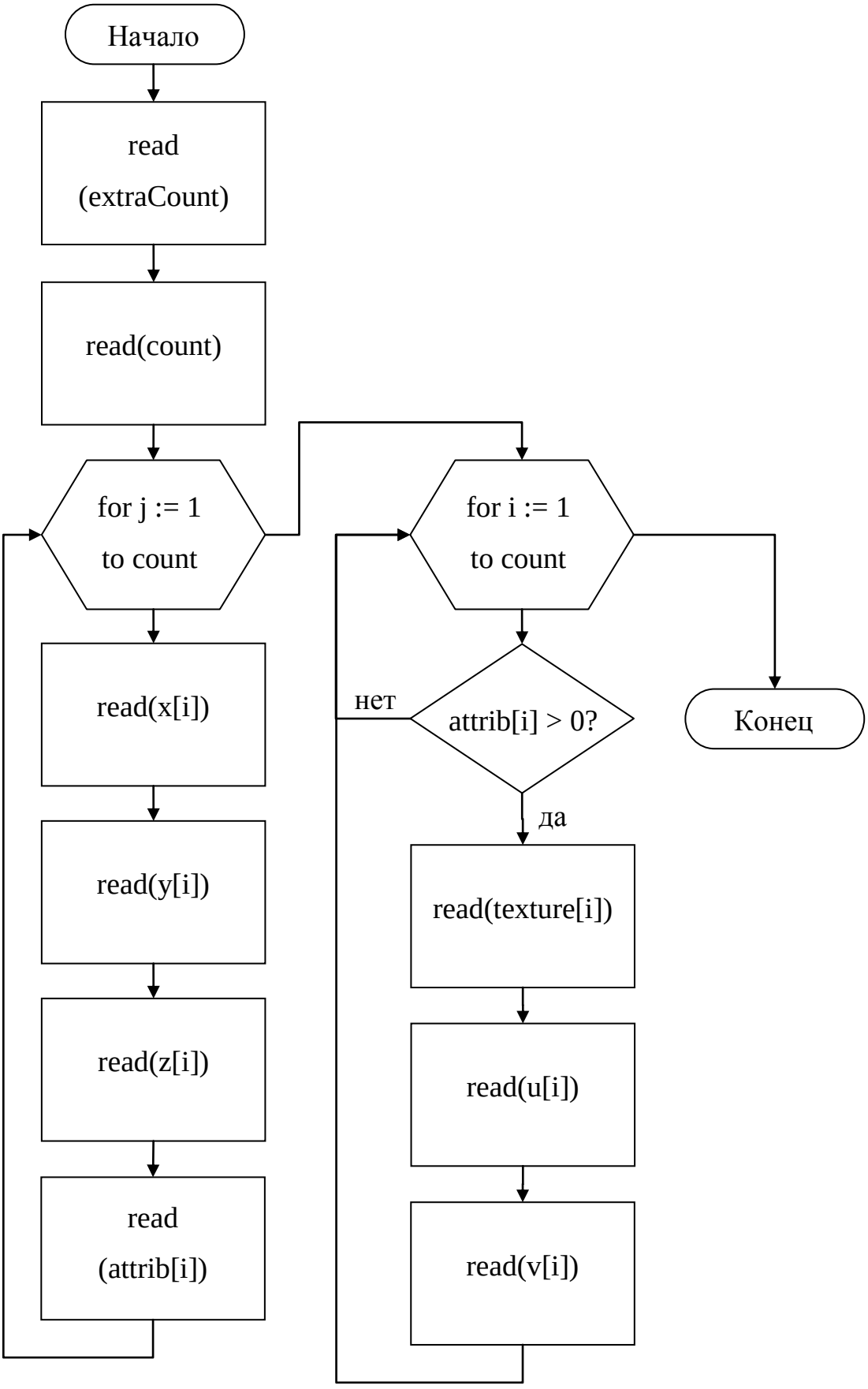
Начало
read
(extraCount)
read(count)
for j := 1
to count
read(x[i])
read(y[i])
read(z[i])
read
(attrib[i])
for i := 1
to count
нет
attrib[i] > 0?
да
read(texture[i])
read(u[i])
read(v[i])
Конец

Рисунок 2.7 – блок-схема чтения расширенных данных

3 АНАЛИЗ КОРРЕКТНОСТИ РЕАЛИЗАЦИИ МОДЕЛИ

3.1 Анализ производительности реализации

В связи с тем, что задачей данной работы является достижение наибольшей производительности, неотъемлемой частью верификации (т. е. проверки разработанной системы на соответствие формальным требованиям) является измерение производительности системы при различных настройках, анализ его изменения и составление выводов на тему эффективности разработанной математической модели.

Перед проведением анализа сформулируем математическую модель испытаний производительности. Она может быть описана формулами ускорения (3.1) и эффективности (3.2) [12].

Коэффициент ускорения (3.1) показывает, во сколько раз увеличится производительность (т. е. увеличится объём обрабатываемых в единицу времени данных, либо уменьшится время обработки фиксированного объёма данных) [16].

$$S_n = \frac{T_1}{T_n} \tag{3.1}$$

где S_n – коэффициент ускорения, действительная безразмерная величина.

T_1 – время работы однопоточного варианта алгоритма, в любых временных единицах;

T_n – время работы параллельной версии алгоритма, работающей на n потоках, в той временной единице, что и T_1;

Эффективность (3.2) показывает, насколько полно задействована система из n вычислительных устройств. При $E_n = 1$ все вычислительные устройства задействованы в полной мере.

$$E_n = \frac{S_n}{n} \tag{3.2}$$

где E_n – эффективность, действительная безразмерная величина;

n – количество потоков.

Цель оптимизации – максимизировать ускорение и приблизить эффективность к единице. Иными словами, найти совместное решение системы оптимизационных задач

$$\begin{cases} S_n \to \max \\ E_n \to 1 \end{cases} \tag{3.3}$$

где S_n – коэффициент ускорения;

E_n – эффективность.

Все нижеприведённые исследования и испытания были проведены с использованием CUDA-совместимого кластера GPGPU NVidia Tesla K10.

Проводимое испытание – это верификация гипотезы о сверхлинейном приросте производительности при увеличении количества фрагментов траектории (иными словами, при увеличении числа частично накрадывающихся траекторий и повторном использовании уже визуализированных фрагментов).

Методика испытания заключалась в следующем.

1. В замкнутом объёме в виде куба размером в 100x100x100 условных единиц случайным образом формировались 10^5 траектории из 10 опорных точек каждая.

2. Затем выполнялось объединение фрагментов маршрута с заданным критерием ε.

3. После этого был произведён подсчёт совокупной длины (в у. е.) пересекающихся участков. Если участок был сформирован из нескольких объединённых фрагментов различных траекторий, то каждый исходный фрагмент вносит свой вклад в итоговую длину фрагмента.

4. В завершении производилась визуализация и замерялось время выполнения программы от момента приёма визуализатором задачи, до момента окончания передачи последнего кадра видеоряда.

Сводные результаты испытаний приведены в табл. 3.1 (в ячейках содержится кадровая частота в кадрах в секунду со множителем 10^5; длины квантованы на 0,5; отобраны длины с наибольшим количеством вхождений). На их основании можно сделать вывод о том, что прирост производительности действительно сверхлинеен; однако установить его точный характер не представляется возможным в связи и большим количеством параметров, влияющих на производительность.

Таблица 3.1 – Сводная таблица производительности реализации

		Пороговое расстояние объединения (ε)			
		0,25	0,5	5,0	10,0
Длина объединённых фрагментов, 10^5 кадров/с	12,0	0,628	0,920	9,328	25,862
	10,0	0,583	0,895	89,630	14,870
	7,5	0,423	0,786	6,258	5,286
	6,0	0,354	0,598	4,186	3,259
	5,0	0,298	0,468	1,100	2,568
	3,5	0,168	0,399	0,795	2,469
	2,0	0,121	0,395	0,626	2,236
	1,0	0,048	0,201	0,452	1,965
	0,5	0,012	0,128	0,416	1,523

3.2 Анализ соответствия реализации и исходной модели

Исходная математическая модель и её реализация обладают следующими важными особенностями:

1. Y-компонента ортонормированного базиса камеры всегда направлена вверх в мировой системе координат.

2. Для интерполяции используется кубический сплайн.

3. Фрагменты траектории, находящиеся друг к другу ближе, чем некоторый порог ε, объединяются. Объединение реализовано как отбрасывание второго и последующего фрагментов траектории и их заменой на первый вариант фрагмента.

Для проверки всех вышеуказанных особенностей были созданы три отладочные траектории, А (см. рис. 3.2), Б (см. рис. 3.3) и В (см. рис. 3.4) соответственно. Траектории А и Б составлены таким образом чтобы иметь совпадающих участок.

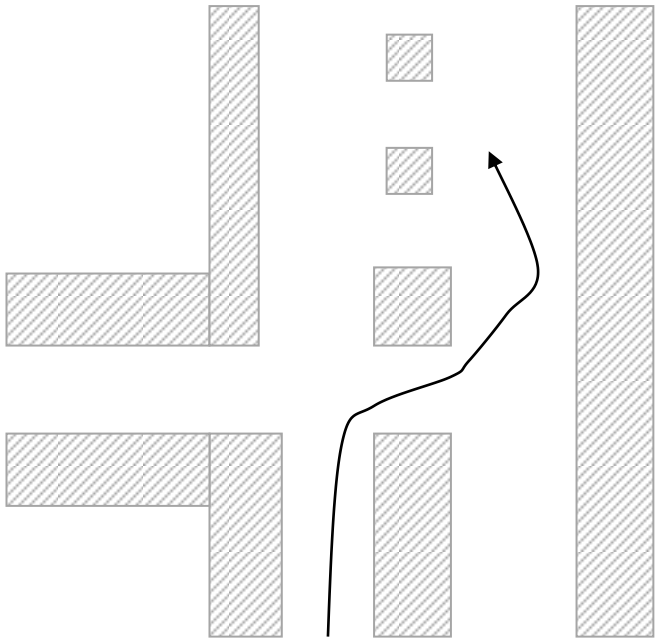

Рисунок 3.2 – испытательная траектория А

Траектория В является усреднённой оптимизированной версией А и Б. Можно отметить, что при $\varepsilon = 0{,}7$ траектория Б была не только визуализирована частично посредством повторного использования фрагмента траектории, но и перестроена – количество и положение узлов на общем участке не совпадает с

таковым ни у траектории А, ни у траектории Б. Также изменилась и скорость прохождения участка из-за изменения расстояния между узлами.

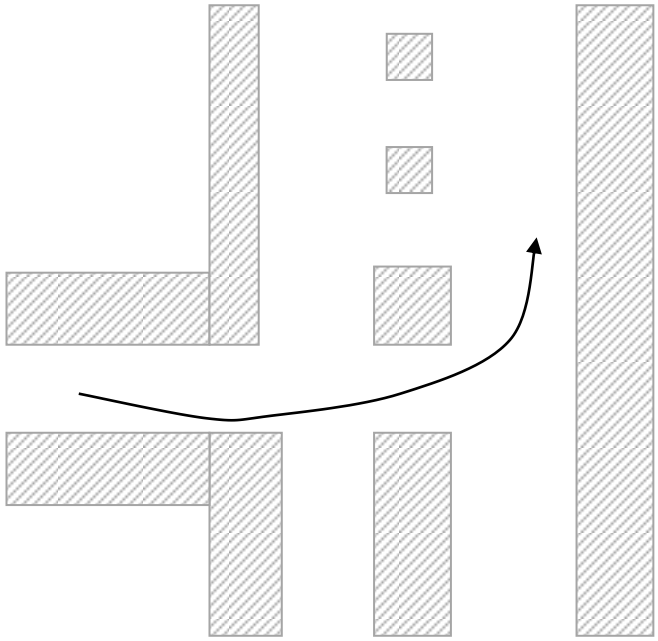

Рисунок 3.3 – испытательная траектория Б

Можно заметить, что изменению подвергся не только общий, но и прилегающий к нему участок. Это подтверждает корректность реализации той части математической модели, которая отвечала за сходимость фрагментов друг к другу.

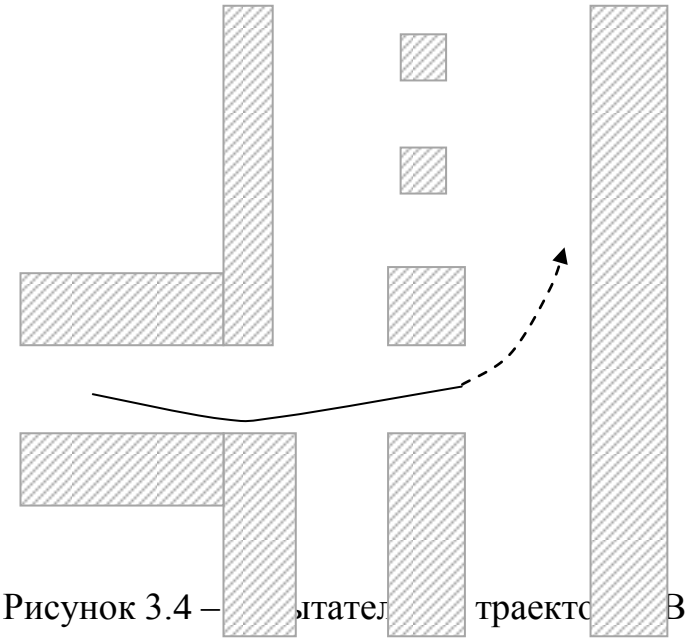

Рисунок 3.4 – ытател траекто В

Во время испытательного запуска была сделана серия экранных снимков с периодичностью в 0,5 секунды с целью демонстрации.

Модель, используемая в данной демонстрации, предоставляется Йельским университетом в неограниченное пользование на безвозмездной основе по лицензии CC-0.

Исходные изображения имели малые контраст и яркость. В целях наглядности данные показатели были искусственно завышены для рис. 3.5–3.7.

Результат визуализации траектории A приведён на рис. 3.5.

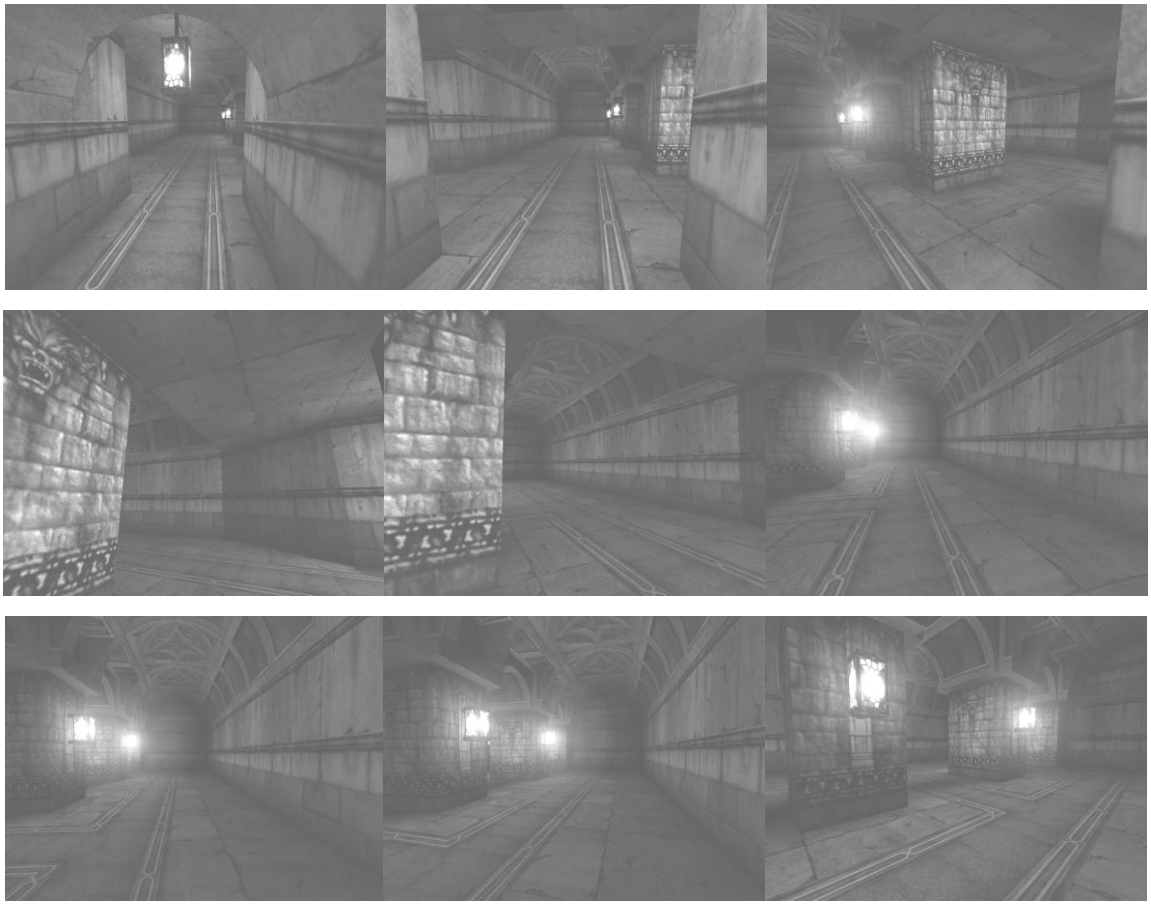

Рисунок 3.5 – визуализация видеоряда для траектории А (рис. 3.2)

Результат визуализации траектории Б приведён на рис. 3.3. Можно отметить, что окончание данной траектории (начиная со снимка №6) схож с конечным участком траектории А (начиная со снимка №4). Стоит отметить, что общим у траекторий А и Б является только общая форма, но не относительное положение контрольных точек.

Результат визуализации траектории В, (рис. 3.4), приведён на рис. 3.7.

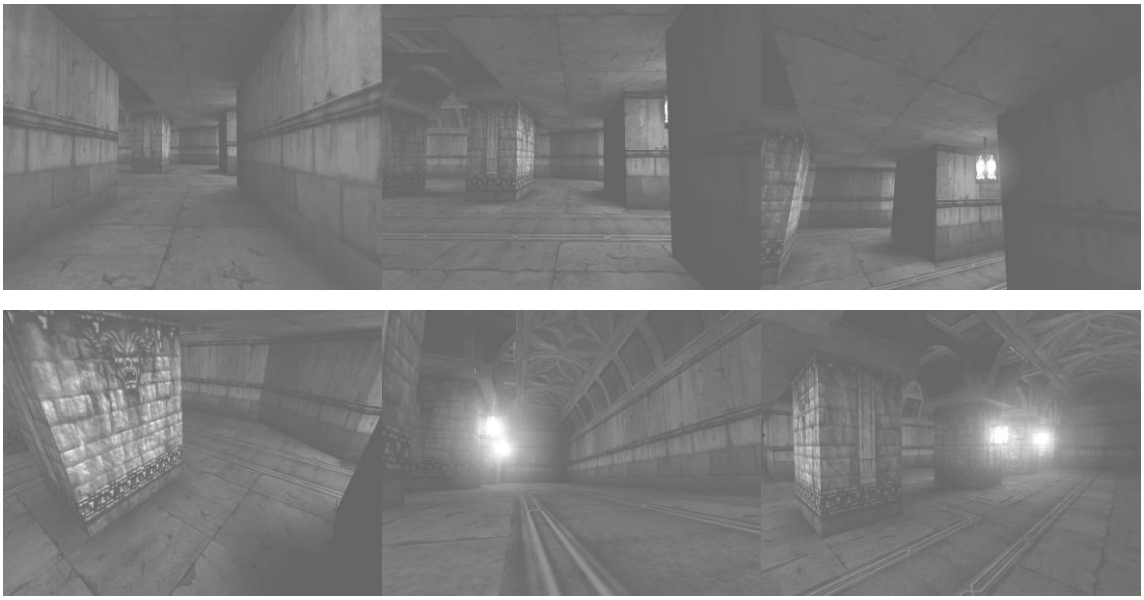

Рисунок 3.6 – визуализация видеоряда для траектории Б (рис. 3.3)

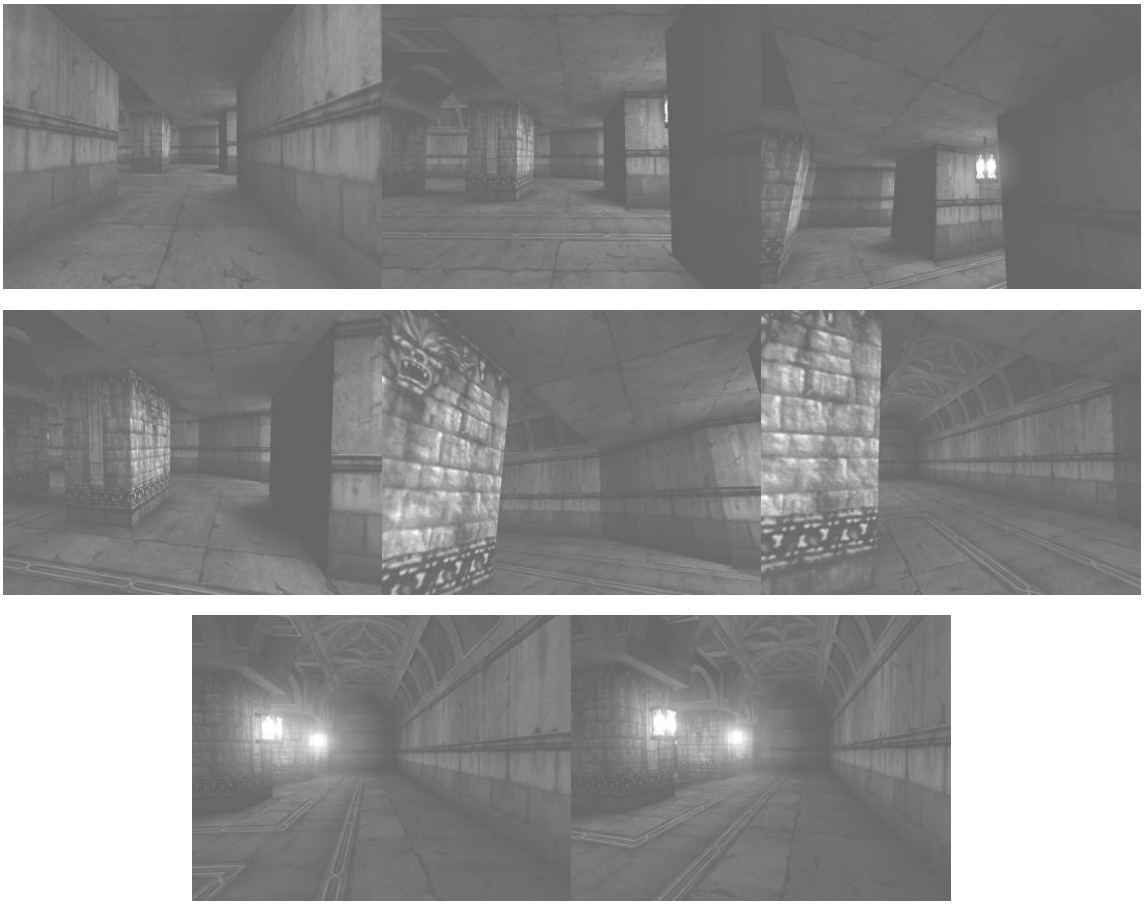

Рисунок 3.7 – визуализация видеоряда для траектории В (рис. 3.4)

Стоит отметить, что данные визуализации служат демонстрацией корректности не только алгоритма оптимизации путей, но также и алгоритма визуализации.

Таким образом, изначальные гипотезы касательно производительности и прогнозируемого характера работы оказались верны. К этим гипотезам относится следующее:

1. При наличии фрагментов различных маршрутов, совпадающих либо проходящих рядом, их объединение с целью повторного использования ранее визуализированного видеоряда способно обеспечить сверхлинейный прирост производительности.

2. При увеличении количества траекторий при неизменном объёме их размещения количество и совокупная длина совпадающих согласно заданному критерию участков также возрастает.

3. Выполнение объединения участков целиком не имеет практического преимущества перед предварительным уточнением границ этих участков в связи с относительно малой длиной задаваемых пользователем сегментов. Малая их длина, в свою очередь, обусловлена сложной геометрией модели (т. е виртуального помещения), требующая частого изменения направления движения виртуальной камеры.

4. Выполнение объединения траекторий способно изменить состав и расположение опорных точек без изменения общей формы интерполирующей кривой.

ЗАКЛЮЧЕНИЕ

В ходе выполнения настоящей выпускной квалификационной работы были сделаны следующие выводы:

1. Результаты настоящей работы имеют большую практическую значимость и могут быть применены в широком спектре сфер человеческой деятельности, включая сферу развлечений, образования, бизнеса, промышленности, медицины, науки.

2. Данная работа актуальна, потенциально востребована и рассматривает нишу, не имеющую полных аналогов. Изученные же аналоги являются частичными и представляют собой сервисы по аренде вычислительной платформы, что находится на уровне ниже проблем и задач, рассматриваемых в данной работе.

3. Была показана возможность составления и реализации комплексной модели, описание которой включает несколько аспектов. В данной работе такими аспектами были: процесс визуализации, расчета траектории движения камеры и повторное использование фрагментов видеорядов. Каждый из них был подкреплён теоретическими выкладками и математическими формулами.

4. Было разработано программное обеспечение, выполняющее оптимизированный вывод по запросу пользователей. Это программное обеспечение можно использовать в составе других программ и программных систем.

5. Благодаря модульности разработанного программного обеспечения, а также унификации и минимизации его программного интерфейса спектр возможного применения является крайне широким и не ограничен рамками клиент-серверной трёхзвенной архитектуры, части которой взаимодействуют посредством сети интернет.

Программное обеспечение, разработанное в рамках данной работы, приложено на электронном носителе (см. приложение И).

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

Научная и методическая литература

1. Архитектура виртуальных миров / Под ред. М.Б. Игнатьева, А.В. Никитина, А.Е. Войскунского. – ГУАП, 2015. – 240 с.

2. Беженцев Р.В. Применение кривых Безье / Р.В. Беженцев // Певзнеровские чтения. – 2013. – № 1. – С. 7-12.

3. Богодухов С.И. Основы проектирования заготовок в автоматизированном машиностроении : учеб. пособие / С.И. Богодухов, А.Г. Схиртладзе, Р.М. Сулейманов, Е.С. Козик. – М. : Машиностроение, 2012. – 432 с.

4. Боресков А. В. и др. Параллельные вычисления на GPU : учеб. пособие / А. В. Боресков, А. А. Харламов ; предисл.: В. А. Садовничий. – М. : Изд-во М. ун-та, 2012. – 336 с.

5. Боресков А. В. Основы работы с технологией CUDA. / А.В. Боресков, А.А. Харламов – М.: ДМК Пресс, 2013. - 232 с: ил.

6. Буймов Б.А. Геометрическое моделирование и компьютерная графика. – М. : ТУСУР, 2011. – 104 с.

7. Власова Е.А. Элементы функционального анализа: учеб. пособие. / Е.А. Власова, И.К. Марчевский. — СПб.: Издательство «Лань», 2015. – 400 с.

8. Вольф Д. OpenGL 4. Язык шейдеров. Книга рецептов. – М. : ДМК Пресс, 2015. – 368 с.

9. Гинсбург Д. OpenGL ES 3.0. Руководство разработчика. / Д. Гинсбург, Б. Пурномо. – М. : ДМК Пресс, 2015. – 448 с.

10. Дэвид В . OpenGL 4. Язык шейдеров. Книга рецептов. Пер. с англ. - М. : ДМК Пресс, 2015. – 480с. – Парал. тит. англ. – ISBN: 978-5-97060-255-3, 978-1-78216-702-0.

11. Зайцев А. В. Информационные системы в профессиональной деятельности : учеб. пособие / А. В. Зайцев, Д. А. Ловцов, С. В. Федосеев ; под

ред. Д. А. Ловцова ; Рос. академия правосудия. – Москва : РАП, 2013. – 179 с. – ISBN 978-5-93916-377-4.

12. Казённов. А.М. Основы технологии CUDA / А.М Казённов // Математические основы и численные методы моделирования. – 2010. – №3. – С. 295–308.

13. Кузнецов, О.П. Дискретная математика для инженера. — СПб. : Лань, 2012. — 400 с.

14. Перепёлкин Е.Е., Садовников Б.И., Иноземцева Н.Г. Вычисления на графических процессорах (GPU) в задачах математической и теоретической физики / Е.Е. Перепёлкин, Б.И. Садовников, Н.Г.Иноземцева. – М. : Ленанд, 2014. – 176 с.

15. Роджерс Д. Алгоритмические основы машинной графики = Procedural Elements for Computer Graphics. – М.: Мир, 2016. – 512 с.

16. Рутм Г. CUDA Fortran для ученых и инженеров. / Г. Рутм, М. Фатика. – М. : ДМК Пресс, 2014. – 364 с.

17. Энтони, У. Параллельное программирование на C++ в действии. Практика разработки многопоточных программ. М. : ДМК Пресс, 2012. — 672 с.

Литература на английском языке

18. Anyuru A. Professional WebGL Programming: Developing 3D Graphics for the Web. – Wrok, 2012 – p. 660.

19. Bankoski J. Technical Overview of VP8, an Open Source Video Codec for the Web / J. Bankoski., P. Wilkins, X. Yaowu. // ACME. –2011. –№7. – p. 21.

20. Blanchard, C. Reality built for two: A virtual reality tool. In Computer Graphics / C. Blanchard et al. // Symposium on Interactive 3D Graphics, R. Riesenfeld and C. Sequin, Eds., vol. 24, 1990. P. 35-36.

21. Bhatotia P. GPU Accelerated Incremental Storage and Computation / P. Bhatotia, R. Rodrigues, A. Verma. // Proc. of the USENIX Conference on File and Storage Technologies, 2012 – P. 86-89.

22. Capps M. NPSNET-V: A New Beginning for Dynamically Extensible Virtual Environments / M. Capps et al. // IEEE Computer Graphics & Applications, vol. 20, 2014. – P. 12-15.

23. Coulouris G. et al. DISTRIBUTED SYSTEMS: Concepts and Design. Fifth Edition. / G. Coulouris, J. Dollimore, T. Kindberg, G. Blair. – Addison-Wesley, 2012. – 1047p.

24. Eberly D. Akima Interpolation for Nonuniform 1D Data. – Geometric Tools. – 2016. – №1. –p. 15-18.

25. Gonzalez R. Digital Image Processing, 2nd Edition. / C. Gonzalez, E. Woods – Addison-Wesley Publishing Company, 2016 – 40 p.

26. Kaner C. Lessons Learned in Software Testing: A Context-Driven Approach 1st Edition. – Addison-Wesley, 2012. – 290с. : ил.

27. Malvar S. Low-complexity Transform and Quantization in H.264/AVC / S. Malvar., A. Hallapuro, M. Karczewicz, L. Kerofsky // IEEE Transactions on Circuit and Systems for Video Technology, Vol. 13, 2014. – No. 7. – , pp. 598-603.

28. Sanders J. CUDA by Example: An Introduction to General-Purpose GPU Programming. / J. Sanders, E. Kandrot – Addison-Wesley, 2012. – 290с. : ил.

29. Sharabayko, M. Contemporary video compression standards: H.265/HEVC, VP9, VP10, Daala / M. Sharabayko, N. Markov // SIBCON. – 2016.– №7. – P.25-28.

30. S. Shi. Implementing Open-Source CUDA Runtime. Real-Time Remote Rendering of 3D Video for Mobile Devices. / Shu S., Won J., Nahrstedt K., Roy H. // MM '09 Proceedings of the 17th ACM international conference on Multimedia, 2014 – P.291-400.

31. Tizon N. ROI based video streaming for 3D remote rendering / N. Tizon, C. Moreno, M. Preda // Proc. IEEE 13th Int. Workshop Multimedia Signal Process. (MMSP), 2011 (10) pp. 1-6.

ПРИЛОЖЕНИЕ А

Сравнительная характеристика существующих программных решений

Таблица. А.1 – Сравнительная характеристика решений

		Аналог		
		Плаза Тольятти	ГМИИ им. Пушкина	Академия культуры Google
Требования	Интерактивность камеры	Нет	Да	Да
	3D	Нет	Фотоснимок 360°	Фотоснимок 360
	Качество изображения	Фотографическое	Фотографическое	Фотографическое
	Средства визуализации	Внешний проигрыватель	Браузер, отображение панорамного фото	Браузер, отображение панорамного фото
	Необходимость физического присутствия	Нет	Нет	Нет (однако количество одновременно смотрящих посетителей ограничено)
	Возможность перемещения камеры	Да	Да (предопределённые точки)	Нет

Важное примечание к данным табл. А.1. Рассматриваемая технология имеет название «видео 360°» и «VR». Настоящая же работа ориентирована на узкую область второй из этих технологий.

ПРИЛОЖЕНИЕ Б

Сравнительная характеристика существующих программных решений

Таблица. Б.1 – сравнительная характеристика различных способов интерполяции

Общий вид	Название	Формула	Кол-во необходимых точек	Особенности
См. рис. 1.3	Линейная	$y = a + bt$	2	Разрывы первой производной на границе сегментов
См. рис. 1.4	Кубическая	$y = a + bt + ct^2 + dt^3$	4	Наличие ложных экстремумов
См. рис. 1.5	Косинусная	$f = \frac{1 - \cos(\pi t)}{2}$ $y = y_i(1 - f) + y_i f$	2	Отсутствие ложных экстремумов; принудительное обнуление производной в точках интерполяции

где a, b, c и d – весовые коэффициенты, вычисляемые по-разному для каждого метода.

t – значение параметра интерполирующего многочлена относительно начала сегмента;

y_i – значение i-й контрольной точки. Учитывая инвариантность интерполирующего полинома относительно конкретного вектора системы

координат, вышеприведённые выкладки применимы ко всем векторам базиса по отдельности и независимо.

ПРИЛОЖЕНИЕ В

Алгоритм расчёта координат точек

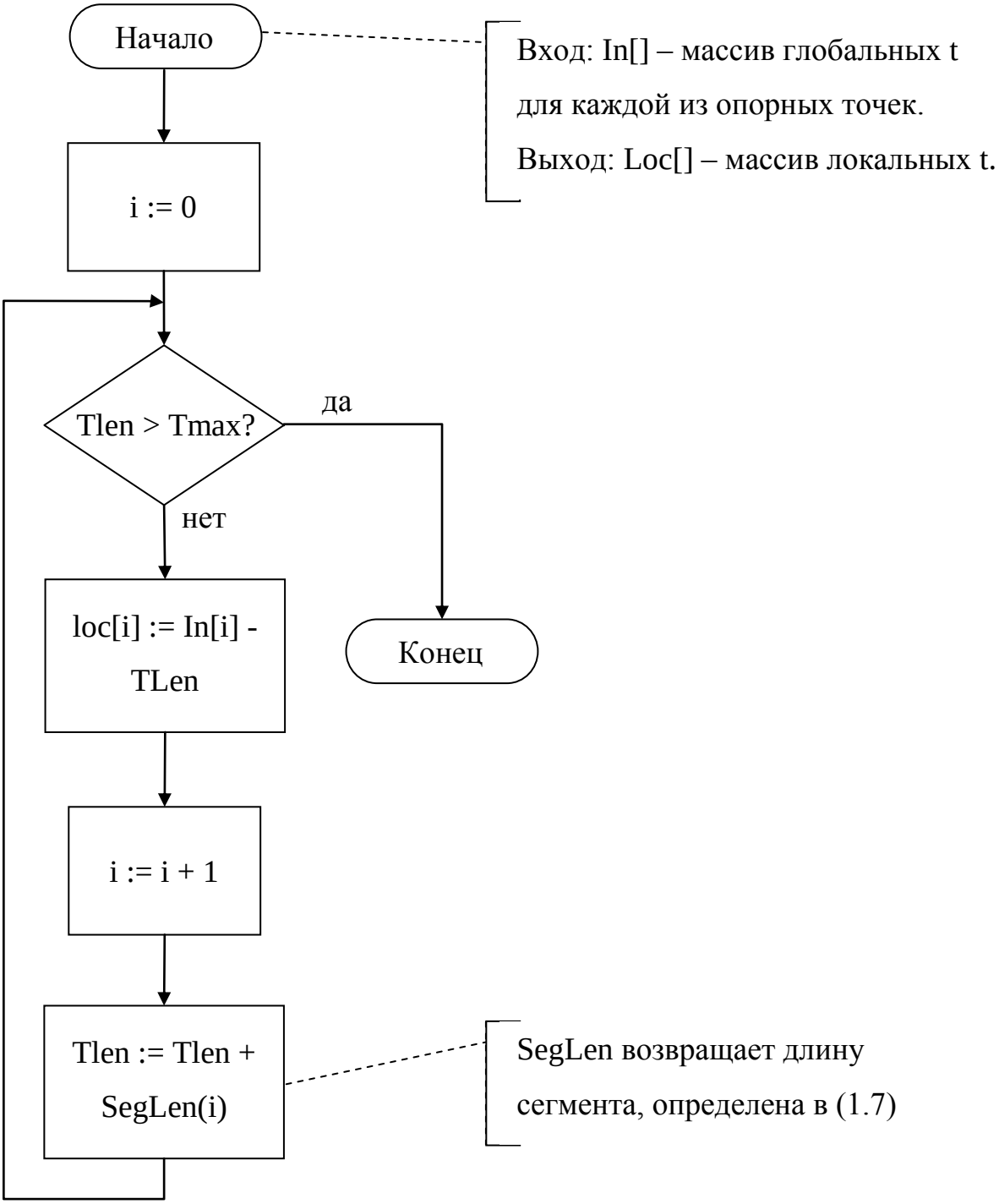

Рисунок В.1 – Блок-схема алгоритма расчета координат точек

ПРИЛОЖЕНИЕ Г

Требования к разрабатываемому программному продукту

Распишем требования по каждому их этих факторов:

1. Функциональные требования:

1.1. Система должна принимать запросы, содержащие траекторию перемещения виртуальной камеры, заданную в виде опорных точек.

1.2. Система должна производить визуализацию видеоряда, перемещая при этом виртуальную камеру согласно переданной траектории.

1.3. В ходе визуализации система должна предоставлять видеоряд как выходные данные.

1.4. Система должна функционировать по графику 24/7/365 (то есть бесперебойно).

1.5. Отказоустойчивость должна обеспечиваться горизонтальным масштабированием (то есть дублированием) всех возможных точек отказа.

1.6. Дубли каждой из точек отказа должны располагаться на физически различных серверах.

1.7. Различные сервера могли бы быть расположены в географически распределённых местах.

1.8. Связность дублей следует обеспечивать балансировщиком пользовательских web-запросов.

1.9. Система не будет использовать механизм Round-Robin DNS (ротации нескольких IP-адресов, назначенных одному доменному имени) [18] .

1.10. Единство балансировщиков не будет обеспечивается механизмом Round-Robin DNS.

1.11. Система должна реализовывать удаленный доступ по аутентификации.

1.12. Система не будет требовать лицензирования.

1.13. Локализацию следует предусмотреть только для пользовательского интерфейса front-end (back-end в локализации не нуждается).

1.14. Процедуры обработки текста должны использовать кодировку Unicode.

1.15. Справочное руководство должно быть ограничить в объёме пошагового руководства для начинающих пользователей.

1.16. Помощь не будет заменена интуитивной организацией интерфейса.

1.17. Почте следует обеспечивать посылку уведомления об окончании расчетов, ведущихся достаточно долгое время. Допустимо использовать сторонних сервисов массовых рассылок во избежание занесения почтового адреса в чёрные списки за распространение незатребованных рассылок (спама).

1.18. Печать не будет требоваться.

1.19. Отчётность не будет требоваться на стороне конечного пользователя.

1.20. Безопасность должна быть организована на уровне мандатного разграничения доступа для защиты ресурсов сервера.

1.21. Сервисный доступ на сервер должен быть организован только с использованием SSL с доступом по закрытому ключу.

1.22. Межсетевой экран должен быть установлен и настроен для защиты от несанкционированных подключений.

1.23. Сторонние организации, специализирующихся на предотвращении и отражении DDOS атак не будут привлечены.

1.24. Системы контроля версий должны использоваться для контроля целостности, резервного копирования и удобства загрузки обновлений.

1.25. Управление системой должно быть организовано удалённый доступ через SSH.

2. Требования к удобству использования:

2.1. Эстетика и логичность пользовательского интерфейса следует реализовать согласно стандартам и распространённым практикам на пользовательский интерфейс.

2.2. Защита от человеческого фактора должна быть реализована через минимизацию доступных привилегий.

2.3. Запрет удаления и замещение должен быть реализован посредством проставления ему пометки «удалёно» вместо реального удаления, с возможностью дальнейшего восстановления.

2.4. Квалификация пользователей и их обучение не будет представлена в виде предварительного обучения вне рамок краткого пошагового руководства.

3. Требования к надёжности:

3.1. Время восстановления после сбоя должно составлять не более 10 минут.

3.2. Ведение журналирования должно производиться в объёме общей работы сервера, а также пользовательских запросов и изменения их статуса в течение всего их жизненного цикла.

4. Требования к производительности:

4.1. Количество одновременно работающих пользователей, которые способна выдержать система, должно составлять не менее 100.

4.2. Время отклика системы (отклик о принятии задания на обработку) должен быть порядка нескольких миллисекунд плюс накладные расходы на передачу данных через интернет.

5. Требования к поддержке:

5.1. Тестируемость следует организовать на высоком уровне за счёт использования подхода TDD (разработка через тестирование) модульной разработки и применения распространённых практик, включая SOLID.

5.2. Расширение не будет производиться в текущей версии.

5.3. Масштабирования следует организовать горизонтальным и свободным за счёт использования Round-Robin DNS, балансировщиков нагрузки и технологии CUDA.

5.4. Адаптация/приспособление к использованию в заданной среде не должна требовать предварительная конфигурация всех компонентов для их корректного сопряжения.

5.5. Совместимость должна быть на уровне стандартов на форматы входных и выходных данных, файлов и потоков. В связи с синхронностью обновления всего программного обеспечения сервера межверсионная совместимость к реализации не планируется.

5.6. Обеспечение:

5.6.1. Сопровождения следует быть непрерывным, на уровне постоянного добавления нового функционала короткими, недельными циклами разработки (является непрерывной частью гибкой методологии разработки, планируемой к внедрению).

5.6.2. Исправления ошибок должно производиться методом непрерывной интеграции.

5.6.3. Обновления данных: при обновлении модуля визуализации все кэши должны очищаться и пересоздаваться по запросу.

5.6.4. Частота архивации и резервного копирования: учётные данные пользователей и их запросы должна быть ежедневно; кэшированные данные резервному копированию не подлежат в связи с возможностью их быстрой регенерации.

5.6.5. Установка должна проводиться через упрощение развёртывания путём создания установочного пакета, способного не только распаковывать файлы и задавать начальную конфигурацию, но и устанавливать дополнительное, стороннее программное обеспечение (например, инфраструктуру CUDA, web-сервер и СУБД с их последующей настройкой под нужды разрабатываемой программы). Изначально планируется сборка APT-

пакетов для Linux-дистрибутива Ubuntu, однако в дальнейшем не исключено создание установочных пакетов иных форматов.

5.6.6. Портируемость не будет организована в связи со стабильностью и предсказуемостью серверной инфраструктуры.

6. Дополнительные требования:

6.1. Ограничения проектирования:

6.1.1. Ограничения на технологии: объекты должны храниться с использованием либо реляционной, либо объектно-ориентированной СУБД.

6.1.2. Процесс разработки следует организовать по Agile (гибкая методология разработки).

6.1.3. Средства разработки: создание диаграмм должно производиться в StarUML, кода – в Qt Creator, хранение кода – в системе контроля версий Git, операционная система – Ubuntu Linux 12.04.

6.2. Ограничения реализации, разработки, построение, написания программного кода:

6.2.1. Язык программирования должен быть C++ стандарта от 2011 года.

6.2.2. Использование библиотек должно быть ограничено распространяемых под свободными лицензиями (то есть одобренными Free Software Foundation).

6.3. Требования к программным интерфейсам:

6.3.1. Взаимодействие с пользователем должно производиться транспортные протоколы: HTTP/1.1, HTTP/2, SPDY и т. Д.. Прикладные протоколы: HTML5, XML, JSON, различные медиаформаты.

6.4. Физические ограничения – в связи с независимостью от платформы явных ограничений на условия эксплуатации аппаратного ограничения не накладывается; необходимо обращение к руководству по эксплуатации.

6.5. Законодательная база.

Закон о персональных данных – категория ИСПДн-О (общедоступные данные).

ПРИЛОЖЕНИЕ Д

Спецификации вариантов использования системы

Таблица. Д.1 – Спецификация варианта использования «Выполнение визуализации»

Имя	Выполнение визуализации
Описание	Создание видеоряда по запросу пользователя
Акторы	Пользователь
Триггеры	Отправление запроса с содержащейся внутри траекторией
Предусловия	1. Траектория должна содержать не менее двух точек.
Основной поток	1. Для каждого переданного сегмента: 1.1. Выполнить среди ранее визуализированных сегментов поиск совпадающего с текущим или близкого к нему. 1.2. В случае удачи выполнить подстановку соответствующего видеоряда в накопительный буфер. 1.3. В случае неудачи: 1.3.1. Вычислить координаты точек на траектории, каждая из которых соответствует положению камеры в пространстве для определённого кадра видеоряда. 1.3.2. Провести визуализацию для каждого из вышеупомянутых кадров в накопительный буфер в порядке следования вычисленных точек. 1.3.3. Поместить визуализированный фрагмент в хранилище для дальнейшего использования в рамках пункта 1.1. 1.3.4. В случае превышения объемом данных в хранилище некоторого порогового объёма выполнить удаление наименее часто используемого фрагмента. 1.4. Послать пользователю содержимое буфера. 1.5. Очистить буфер.
Постусловия	Хранилище имеет не менее одного фрагмента и ассоциированного с ним видеоряда.
Альтернативные	Нет

потоки	

ПРИЛОЖЕНИЕ Е

Диаграмма развёртывания программно-аппаратного комплекса удалённой визуализации и доставки видеопотока

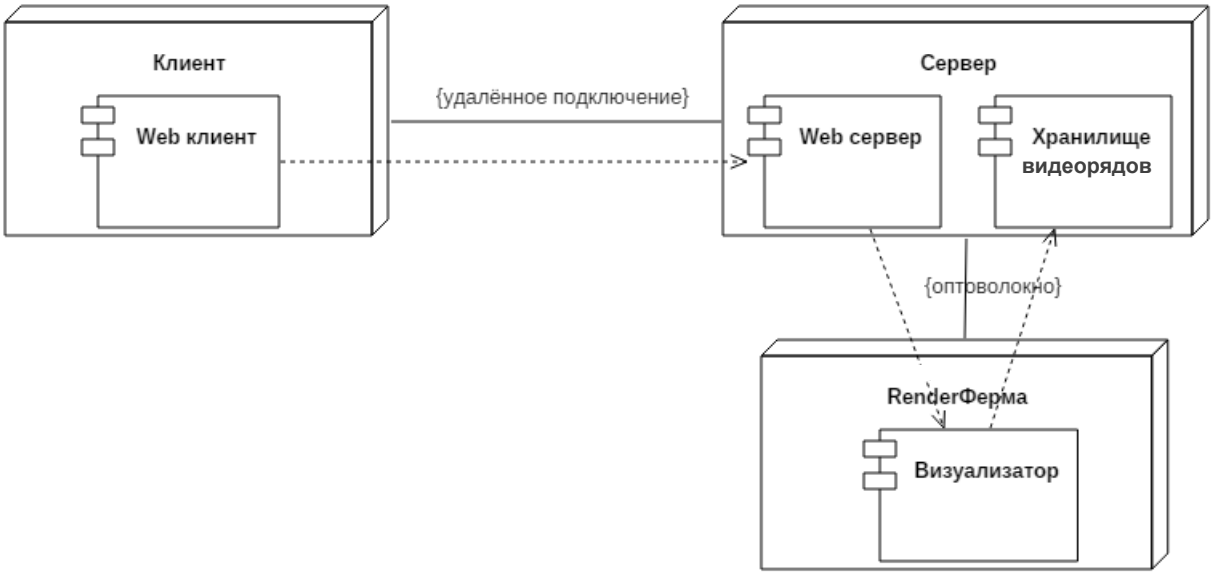

Рисунок Е.1 – Диаграмма развёртывания

На рис. Е.1 приведена типовая структура удалённого визуализатора, являющегося частью трёхзвенной архитектуры распределённых систем.

ПРИЛОЖЕНИЕ Ж

Блок-схема алгоритма построения интерполирующего многочлена

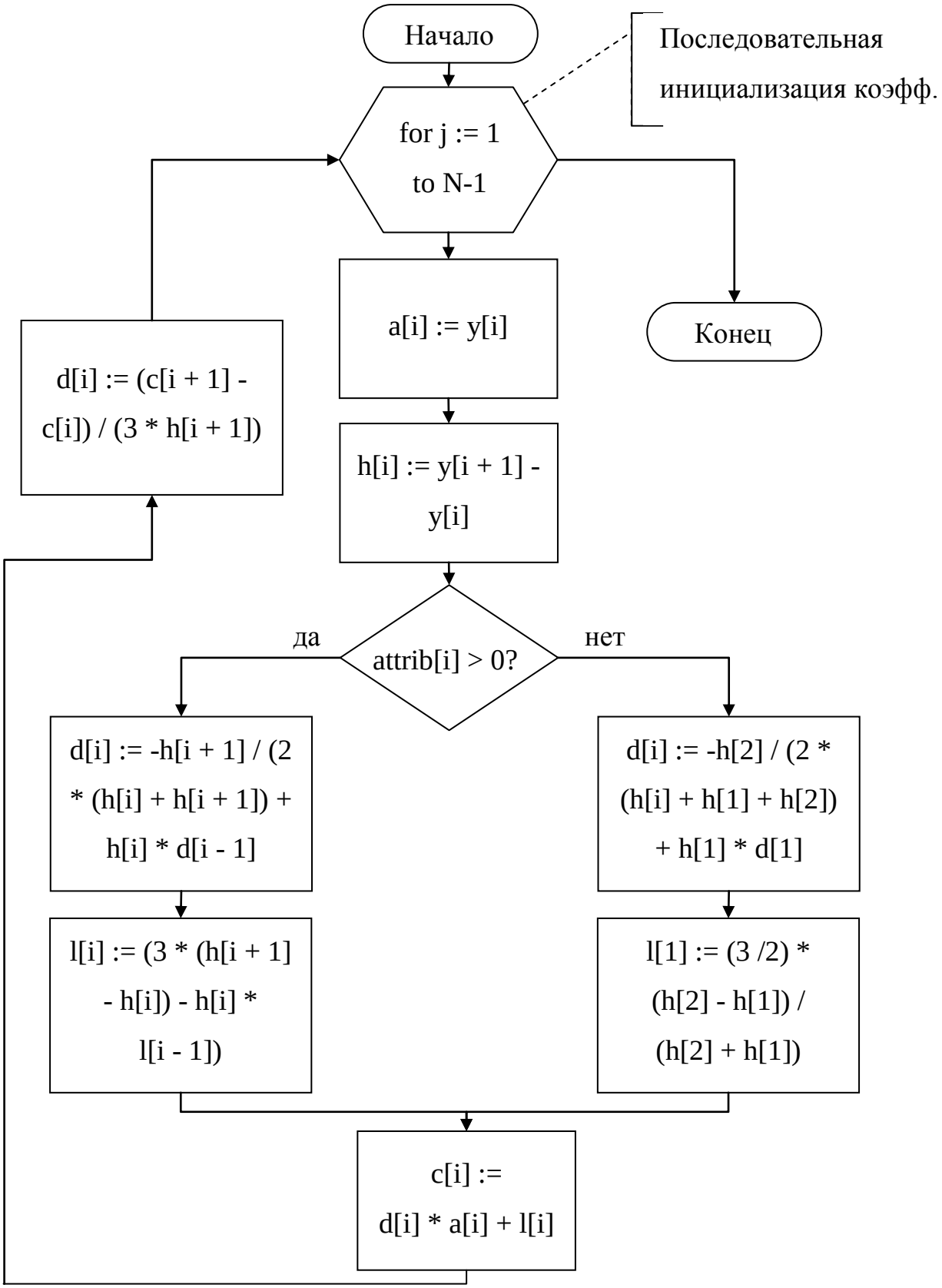

Рисунок Ж.1 – Блок-схема построения многочлена

ПРИЛОЖЕНИЕ И

Носитель с исходным кодом и исполняемыми файлами программы