Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:
«Программа для создания полигональной модели по томографии трехмерного объекта»

Студент ИУ7-53Б (Группа) _______________ (Подпись, дата) А. И. Дегтярев (И. О. Фамилия)

Руководитель курсовой работы _______________ (Подпись, дата) А. А. Павельев (И. О. Фамилия)

2021 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

С продвижением прогресса появляется необходимость в инструментах, находящихся на пересечении нескольких наук. Одним из таких пересечений является медицина и компьютерная графика. С помощью магнитно-резонансной томографии и компьютерной томографии собирают информацию о внутренней структуре органов и тканей. Затем эти данные должны быть корректно отображены на экране компьютера, чтобы медицинский специалист смог поставить диагноз.

В то же время, сейчас активно развиваются и применяются технологии трехмерной печати. В медицине они используются для создания протезов и имплантов. Индивидуально напечатанные протезы значительно увеличивают качество жизни. Но для их создания необходима информация о внутренней структуре заменяемого органа. Таким образом, возникает потребность в программном обеспечении, позволяющем по трехмерным снимкам получать файлы для трехмерной печати.

Целью моей работы реализация программы с пользовательским интерфейсом для создания полигональной модели по результатом томографии.

Для достижения поставленной цели необходимо выполнить следующие задачи:

- изучить и проанализировать алгоритмы компьютерной графики построения полигональных моделей из послойных снимков, выбрать наиболее подходящий алгоритм;
- подробно изучить выбранный алгоритм для применения к поставленной задаче;
- привести схему рассматриваемого алгоритма;
- описать используемые структуры данных;
- описать структуру разрабатываемого программного обеспечения;
- определить средства программной реализации;
- описать процесс сборки приложения;
- протестировать разработанное программное обеспечение;

1 Аналитический раздел

В данном разделе будут рассмотрены алгоритмы компьютерной графики построения полигональных моделей из послойных снимков и выбраны наиболее подходящие алгоритмы.

1.1 Формализация объектов

Томография - получение послойного изображения внутренней структуры объекта (см. рис. 1.1)

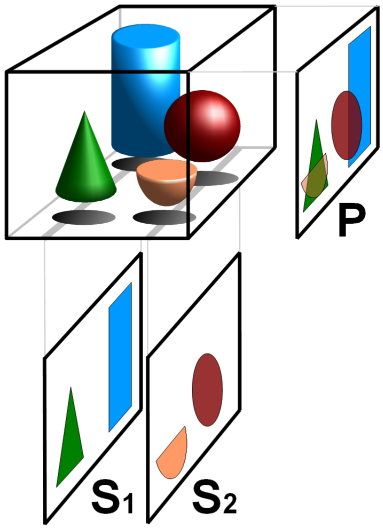


Рисунок 1.1 – Томограммы (S1, S2) группы трёхмерных объектов и их проекция (P)

Результат томографии - регулярная сетка вокселей, в которой каждому вокселю соответствует усредненное значение (температура, плотность материала) в данной точке трехмерного объекта. Данные результата томографии идеально подходят для хранения в трехмерном массиве.

Изоповерхность - поверхность, представляющая точки с постоянным значением (например, плотности, давления, температуры, или скорости) в некоторой части пространства. Математически определяется как поверхность уровня:

$$L_c(f) = \{(x, y, z) | f(x, y, z) = c\} \tag{1.1}$$

Полигональная сетка - совокупность вершин, ребер и граней, которые определяют форму многогранного объекта.

1.2 Анализ алгоритмов полигонизации

Алгоритмы полигонизации разделяют на primal (прямые) и dual (двойственные). Dual алгоритмы в большинстве являются более поздними и способны более точно передать грани поверхности. Основное отличие их в том, как они строят полигоны из регулярной сетки значений функции. Primal ставят вершины на ребрах сетки, а dual определяют позицию вершины внутри куба сетки. Если представить узлы регулярной сетки вершинами графа, то можно сказать, что primal строят полигоны используя прямой граф, а dual двойственный к нему. На рисунке 1.2 показана разница между алгоритмами.

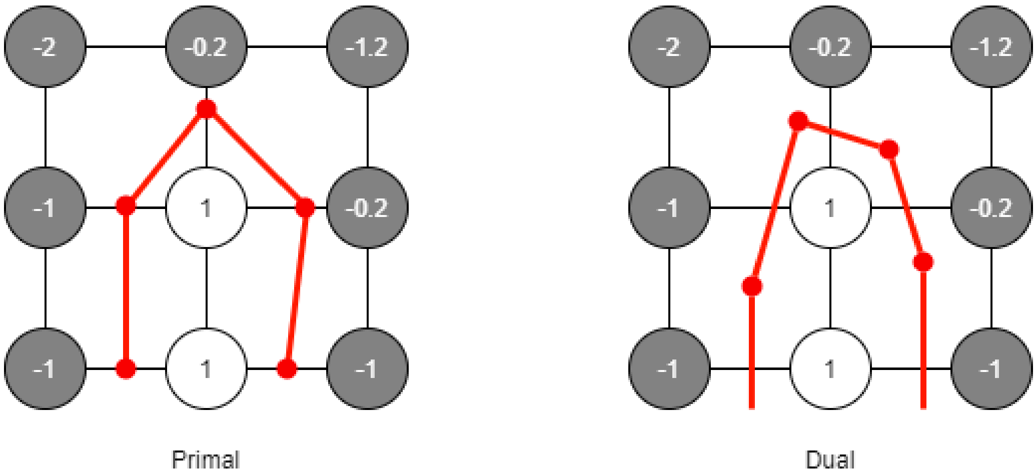


Рисунок 1.2 – Красная линия – аппроксимация изолинии отрезками. В primal алгоритмах вершины отрезков расположены на ребрах сетки, а в dual алгоритмах – внутри ячеек сетки

1.2.1 Алгоритм Marching cubes ("шагающих кубов")

Алгоритм шагающих кубов [1] рассматривает каждый куб в регулярной сетке, анализирует значения в вершинах куба, и определяет необходимые для представления части изоповерхности полигоны, проходящей через данный куб. Так как алгоритм выбирает полигоны, исходя только из положения вершин куба относительно изоповерхности, всего получается 256 (2^8) возможных конфигураций полигонов, которые можно вычислить заранее и сохранить в массиве.

Этот алгоритм был опубликован первее остальных и является основным, наиболее часто используемым. Однако, он создает большое количество поли-

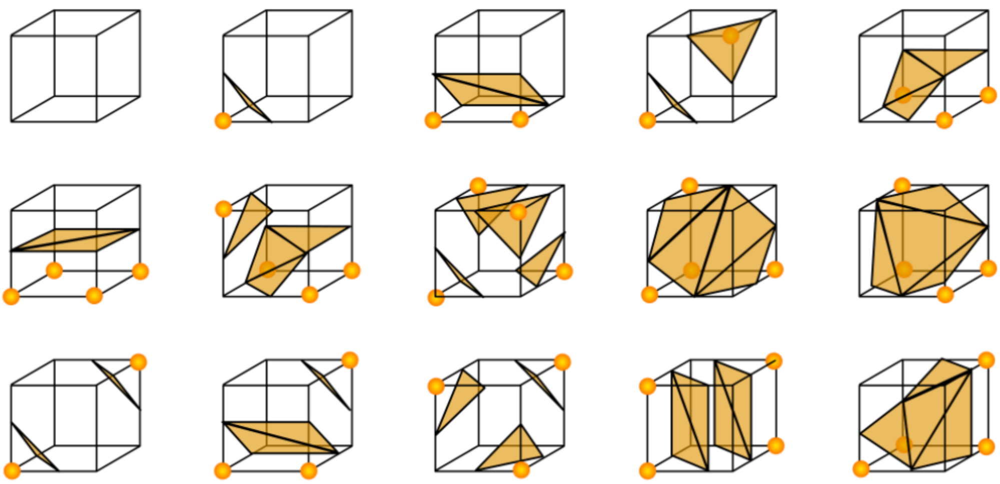


Рисунок 1.3 – 256 возможных конфигураций куба могут быть сведены к 15 благодаря симметрии

гонов - от 1 на каждый пересекающий изоповерхность куб. Это значительно увеличивает нагрузку на отрисовку и какое-либо редактирование. Углы получаются сглаженными. Существуют случаи с неопределенностью (см. рис. ??), который необходимо разрешать дополнительной проверкой соседних вершин.

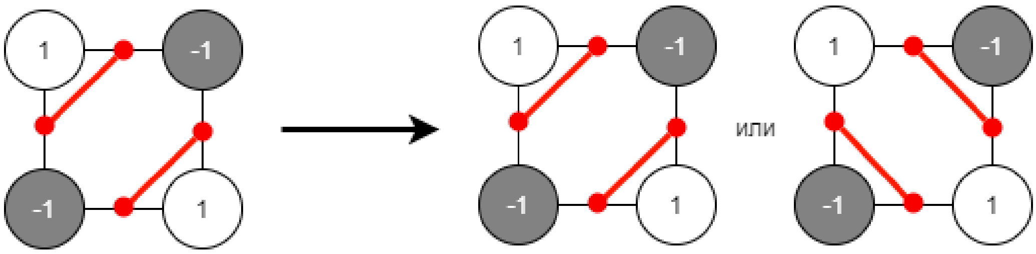


Рисунок 1.4 – Случай неопределенности в двумерном варианте

1.2.2 Алгоритм Marching Tetrahedra

Алгоритм Marching Tetrahedra[2] решает случай неопределенности из предыдущего алгоритма путем разбиения куб на тетраэдры (см. рис. 1.5). Тетраэдр имеет 16 (2^4) возможных конфигураций полигонов.

Изначально этот алгоритм был придуман, как открытый аналог Marching Cubes, так как второй был запатентован. Однако, в наше время патент на Marching Cubes закончился.

1.2.3 Алгоритм Dual Contouring

Алгоритм Dual Contouring[3] решает проблемы Marching Cubes с помощью градиента функции. Эта информация позволит создавать острые

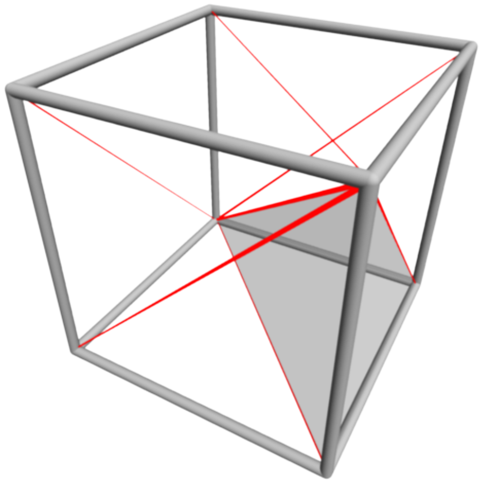


Рисунок 1.5 – Куб, составленный из 6 тетраэдров

границы, разрешать неопределенные случаи. Dual Contouring помещает в каждую ячейку по одной вершине, а затем соединяет точки, создавая полигональную сетку. Точки соединяются вдоль каждого ребра, имеющего смену знака, как и в Marching Cubes. На рисунке 1.6 представлен пример работы алгоритма.

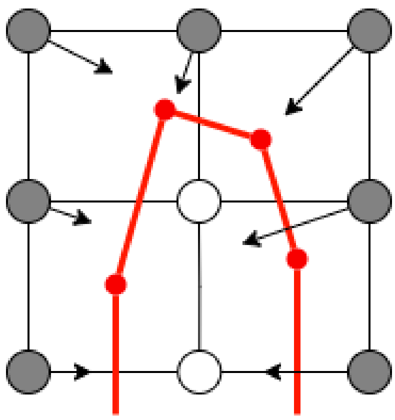


Рисунок 1.6 – Пример работы алгоритма Dual Contouring

На рисунке 1.6 в вершинах сетки показаны направления градиентов функции. Вершины внутри ячеек расставляются таким образом, что построенная изолиния должна быть перпендикулярна этим градиентам

Dual Contouring создаёт более естественные формы, чем Marching Cubes и не требует создания таблицы конфигураций.

1.2.4 Алгоритм Dual Marching Cubes

Алгоритм Dual Marching Cubes[4] является некоторой комбинацией Marching cubes и Dual Contouring. Его главной особенностью считается построение полигональной сетки без избыточного деления на полигоны. Это достигается благодаря построению восьмеричного дерева, в котором куб делится на 8 меньших кубов. В результате разбиения в каждом кубе существует такая точка, касательная плоскость к которой наилучшим образом описывает участок изоповерхность внутри этого куба. С помощью дерева каждая вершина соединяется ребром с соседними к ней. Далее, рассматривая полученный граф, как регулярную сетку, применяют алгоритм Marching Squares. На рисунке 1.7 приведено сравнение алгоритмов полигонизации[4] на примере комнаты , построенной с использованием конструктивной блочной геометрией.

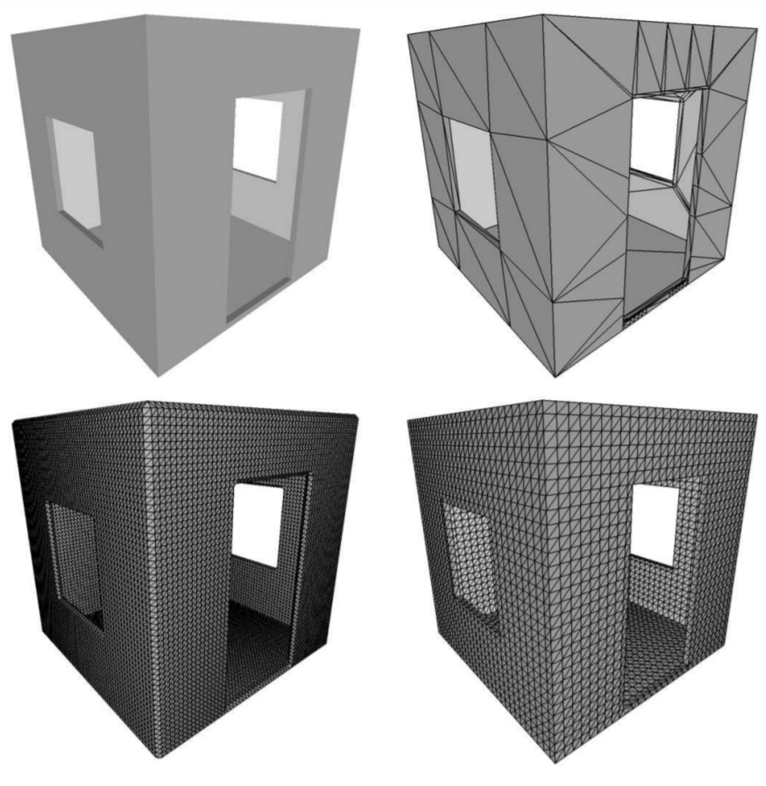


Рисунок 1.7 – Слева вверху - комната-оригинал, построенная с использованием конструктивной блочной геометрией. Слева снизу - результат работы Marching Cubes (67 тыс. полигонов). Справа снизу - Dual Contouring (17 тыс. полигонов). Сверху справа - Dual Marching Cubes (440 полигонов)

В результате работы алгоритма получается полигональная сетка с меньшим количеством полигонов и большей детализацией.

Вывод

В данном разделе были изучены и проанализированы алгоритмы полигонизации. В качестве решения были выбраны алгоритмы Dual Marching Cubes, потому что создает наиболее оптимизированную полигональную сетку, не теряющую деталей.

2 Конструкторский раздел

В данном разделе будут приведена схема алгоритма Dual Marching Cubes, описаны используемые структуры данных, оценен объем памяти, необходимый для хранения данных, описана структура разрабатываемого программного обеспечения.

2.1 Схема алгоритма Dual Marching Cubes

На рисунках 2.1-2.8 представлена схема алгоритма Dual Marching Cubes.

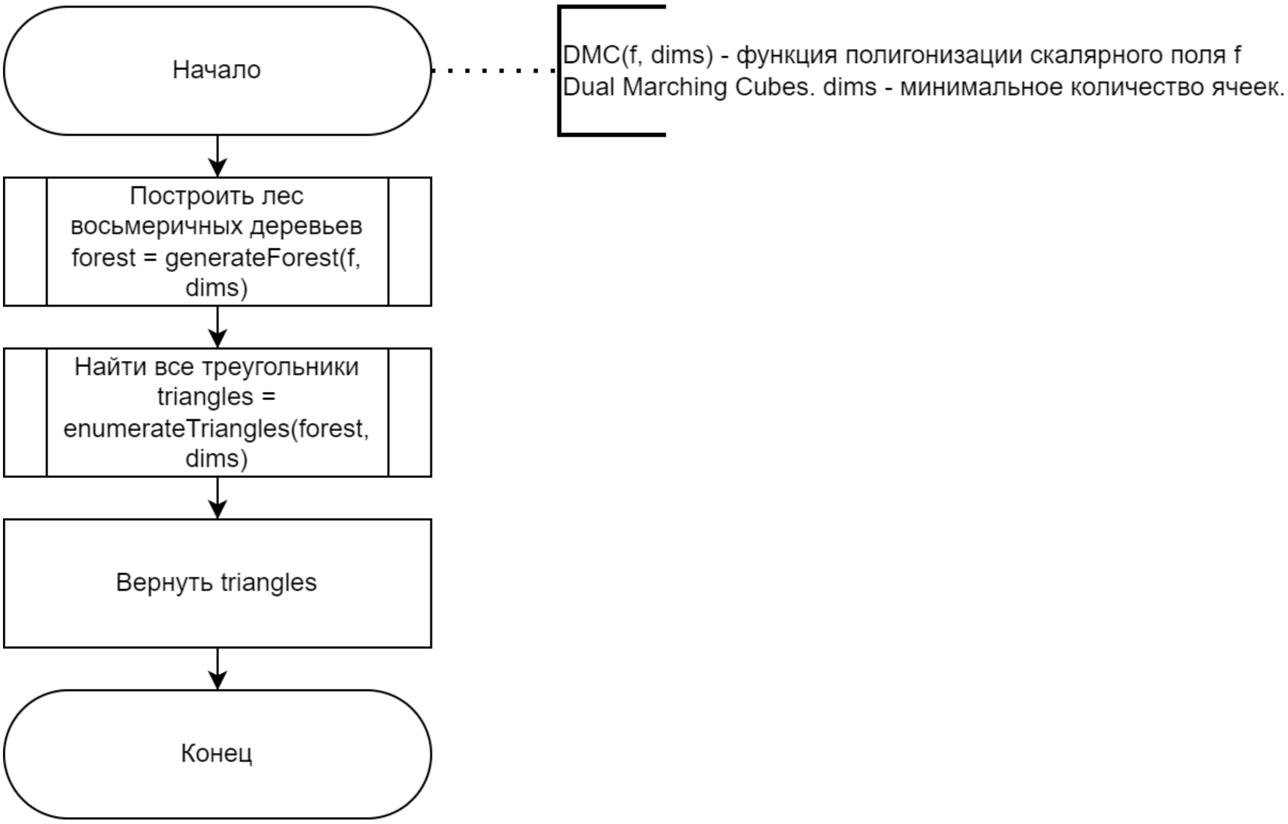


Рисунок 2.1 – Схема алгоритма Dual Marching Cubes

На рисунке 2.1 видно, что алгоритм разделен на 2 этапа:

- 1 этап - построение леса восьмеричных деревьев, характеризующих функцию скалярного поля.
- 2 этап - нахождение всех треугольников, описывающих изоповерхность.

2.1.1 Построение леса восьмеричных деревьев

Согласно работе[4] построение леса восьмеричных деревьев позволяет оптимизировать последующее построение полигональной модели. В каждом листе дерева определяется точка (x, y, z) характеризующая особенность скалярного поля в объеме ячейки, за которую отвечает данный лист дерева. Для этого используется квадратичная функция ошибки:

- рассматривается регулярная сетка внутри ячейки P:

$$P = (x, y, z)|x = x_1 + d_x i y = y_1 + d_y j z = z_1 + d_z k, \tag{2.1}$$

  где (x_1, y_1, z_1) - угол ячейки; $i, j, k \in \overline{0..N}$; d_x, d_y, d_z - шаг сетки;

- в каждой точки $(x_i, y_i, z_i) \in P$ строится тангенсальная плоскость к графику функции в точке $w = f(x, y, z)$, и имеет уравнение:

$$T_i(x, y, z) = w \tag{2.2}$$

$$T_i(x, y, z) = \nabla f(x_i, y_i, z_i) \cdot ((x, y, z) - (x_i, y_i, z_i)); \tag{2.3}$$

- квадратичная функции ошибки для всех точек имеет вид:

$$E(w, x, y, z) = \sum_i \frac{(w - T_i(x, y, z))^2}{1 + |\nabla f(x_i, y_i, z_i)|^2}. \tag{2.4}$$

Точка, в которой квадратичная ошибка минимальна, аппроксимирует функцию наилучшим образом в данном объеме.

Для поиска минимума функции 2.4 воспользуемся методом Якоби для решения системы линейных уравнений, в котором каждое уравнение имеет вид:

$$\nabla f(x_i, y_i, z_i) \cdot (x, y, z) - w = \nabla f(x_i, y_i, z_i) \cdot (x_i, y_i, z_i) \tag{2.5}$$

Однако использовать такие уравнения не результативно - координаты полученной точки часто находятся вне объема ячейки, поэтому нужно добавить ограничение. Сместим точку отсчета координат для точек (x_i, y_i, z_i) в центр ячейки m и добавим уравнения, которые будут устремлять результат в новое начало координат. Теперь система уравнений выглядит следующим образом:

$$\nabla f(x_i, y_i, z_i) \cdot (x, y, z) - w = \nabla f(x_i, y_i, z_i) \cdot ((x_i, y_i, z_i) - m) + f(x_i, y_i, z_i) \tag{2.6}$$

$$c_x x = 0 \tag{2.7}$$

$$c_y y = 0 \tag{2.8}$$

$$c_z z = 0 \tag{2.9}$$

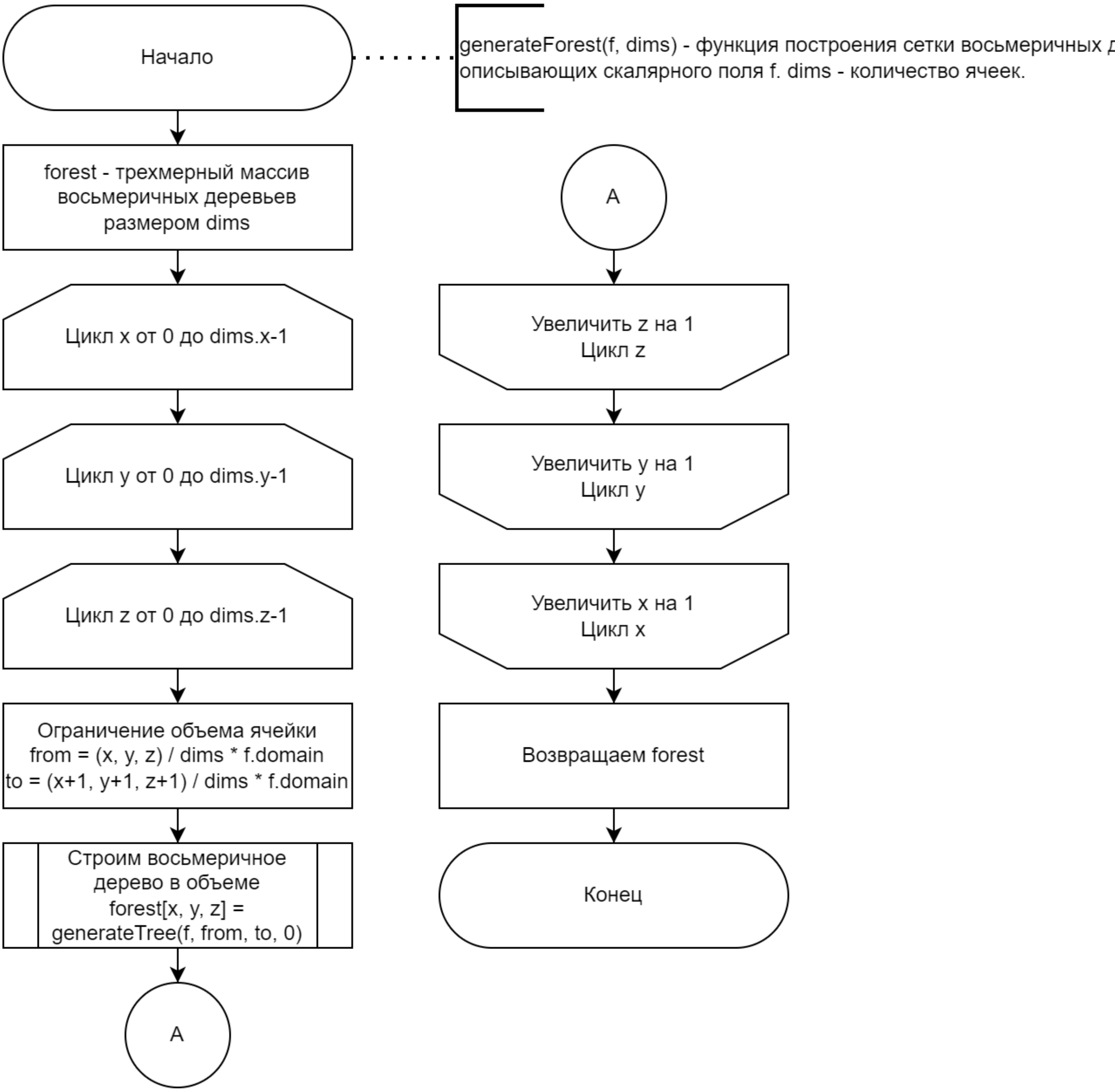


Рисунок 2.2 – Схема первого этапа алгоритма, функции `generateForest` построения леса восьмеричных деревьев

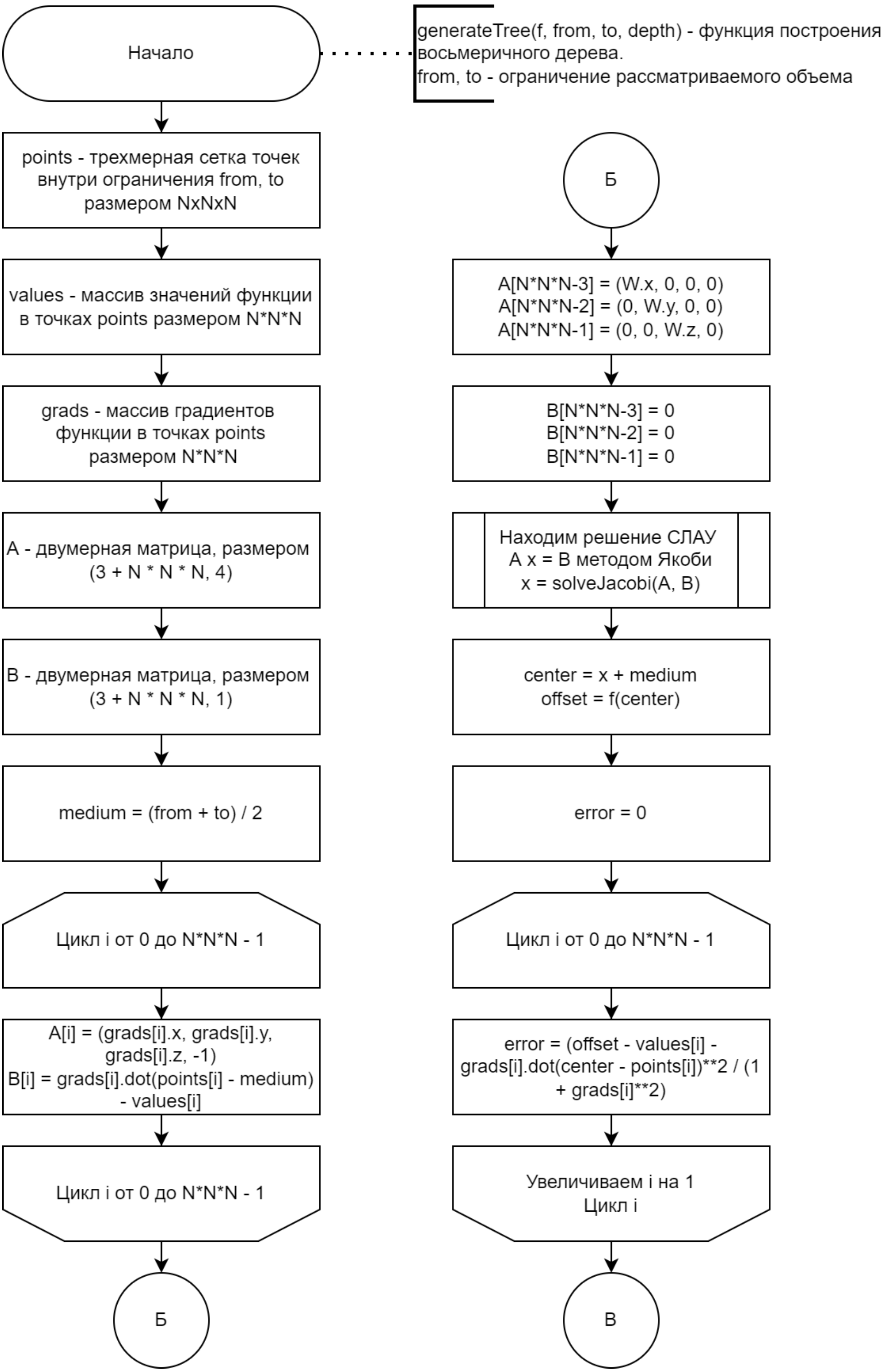


Рисунок 2.3 – Схема функции `generateTree` построения восьмеричного дерева, характеризующего функцию скалярного поля в параллелепипеде

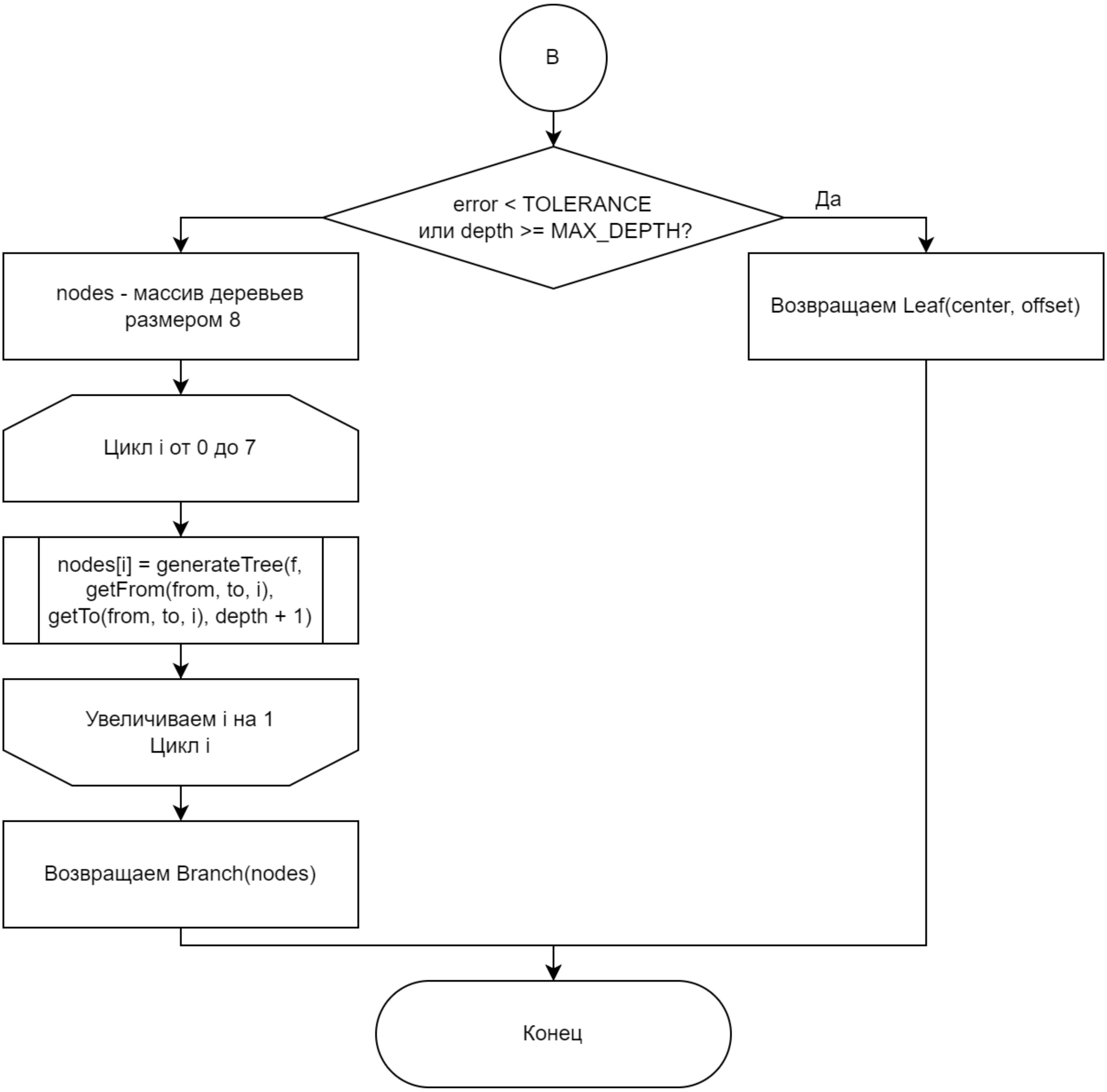


Рисунок 2.4 – Продолжение схемы функции `generateTree` построения восьмеричного дерева. Поиск точки, характеризующей функцию в данном объеме

2.1.2 Нахождение всех треугольников изоповерхности

На рисунках 2.6-2.8 представлен второй этап алгоритма, суть которого заключается в обходе всех соседних ячеек восьмеричных деревьев и применение к соседям алгоритма Marching Cubes. В статье[4] предлагают рекурсивное решение этой задачи на двумерном примере(см. рис. 2.5):

- к каждой узлу дерева применяют `enumerateCell`. Если ячейка является поддеревом, к каждому дочернему узлу функция применяет `enumerateCell`, к каждым парам применяет `enumerateEdgeX` или `enumerateEdgeY` и ко всем узлам применяет `enumerateVertex`;

- `enumerateEdgeX` применяется к паре узлов, имеющих общую сторону по оси X. Если хотя бы один из узлов является поддеревом, то функция применяет `enumerateEdgeX` и `enumerateVertex` ко всем ближайшим к стороне узлам. Аналогично работает `enumerateEdgeY`;

- `enumerateVertex` применяется к четырем узлам, имеющим общую точку. Если хотя бы один из узлов является поддеревом, то применяет ко всем ближайшим к точке `enumerateVertex`. Если же все узлы являются листьями, применить к вершинам в этих листьях Marching Cubes.

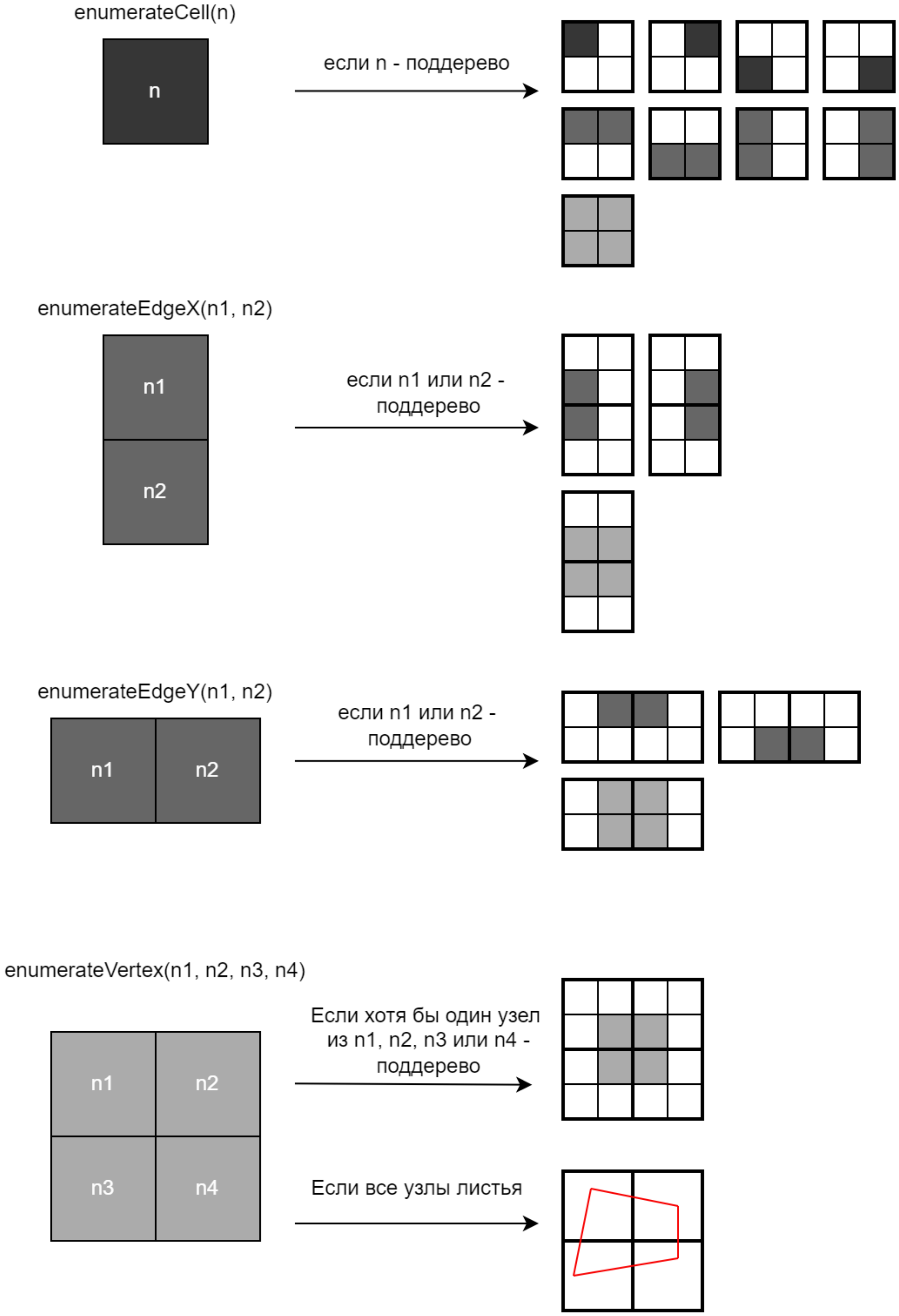


Рисунок 2.5 – Схема рекурсивного обхода соседей, для двумерного варианта. Трехмерный вариант является тривиальными расширением двумерного

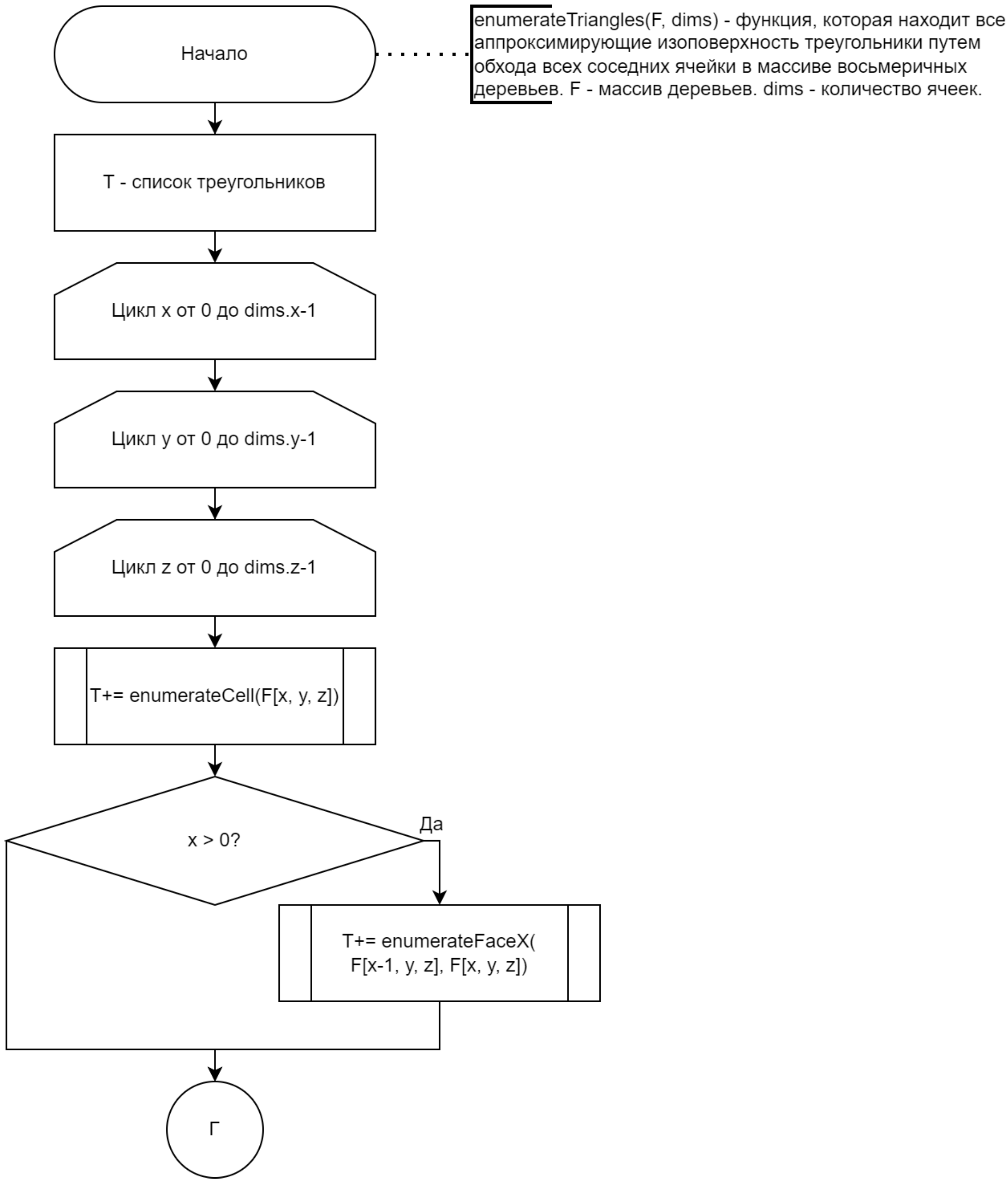


Рисунок 2.6 – Схема функции `enumerateTriangles`

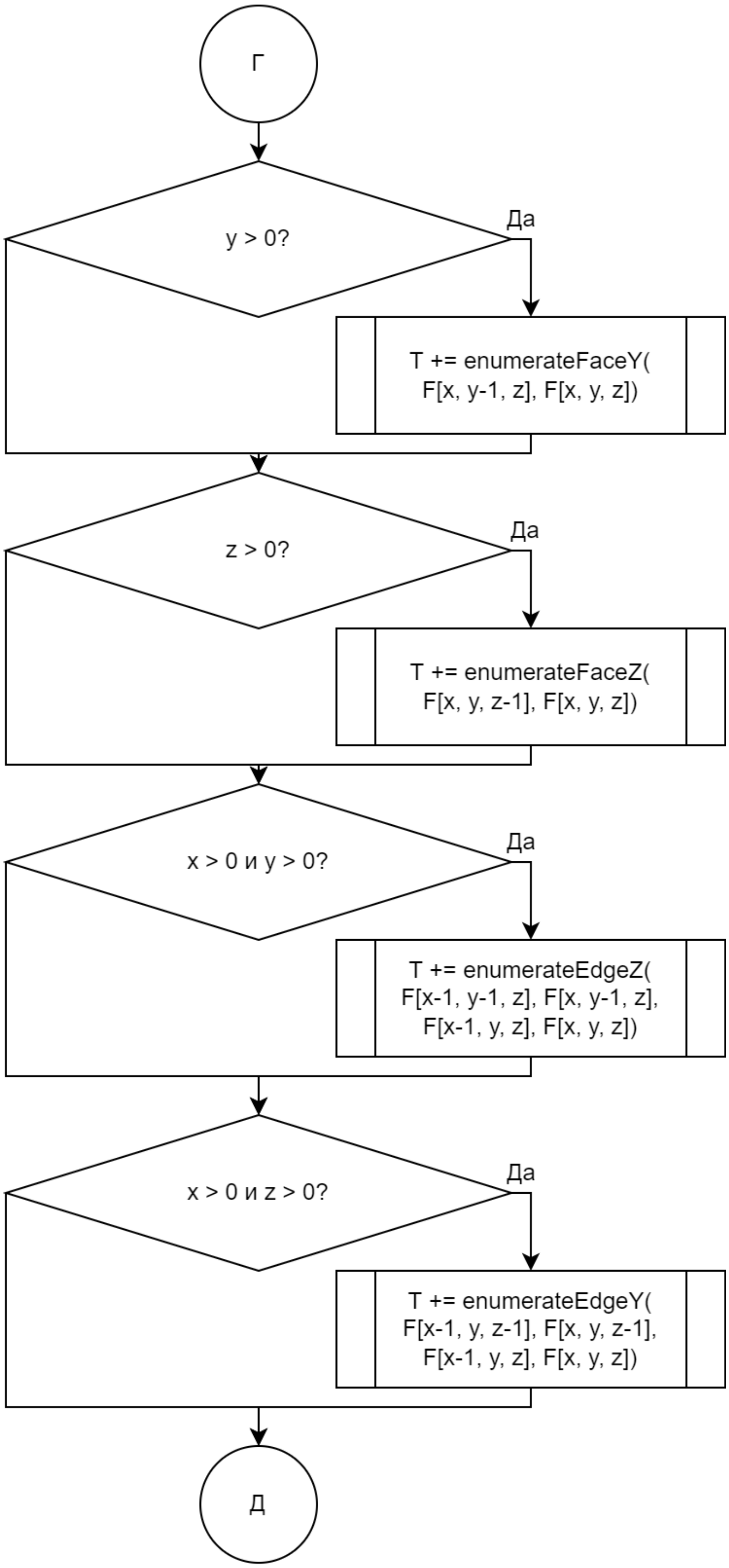


Рисунок 2.7 – Продолжение схемы функции `enumerateTriangles`

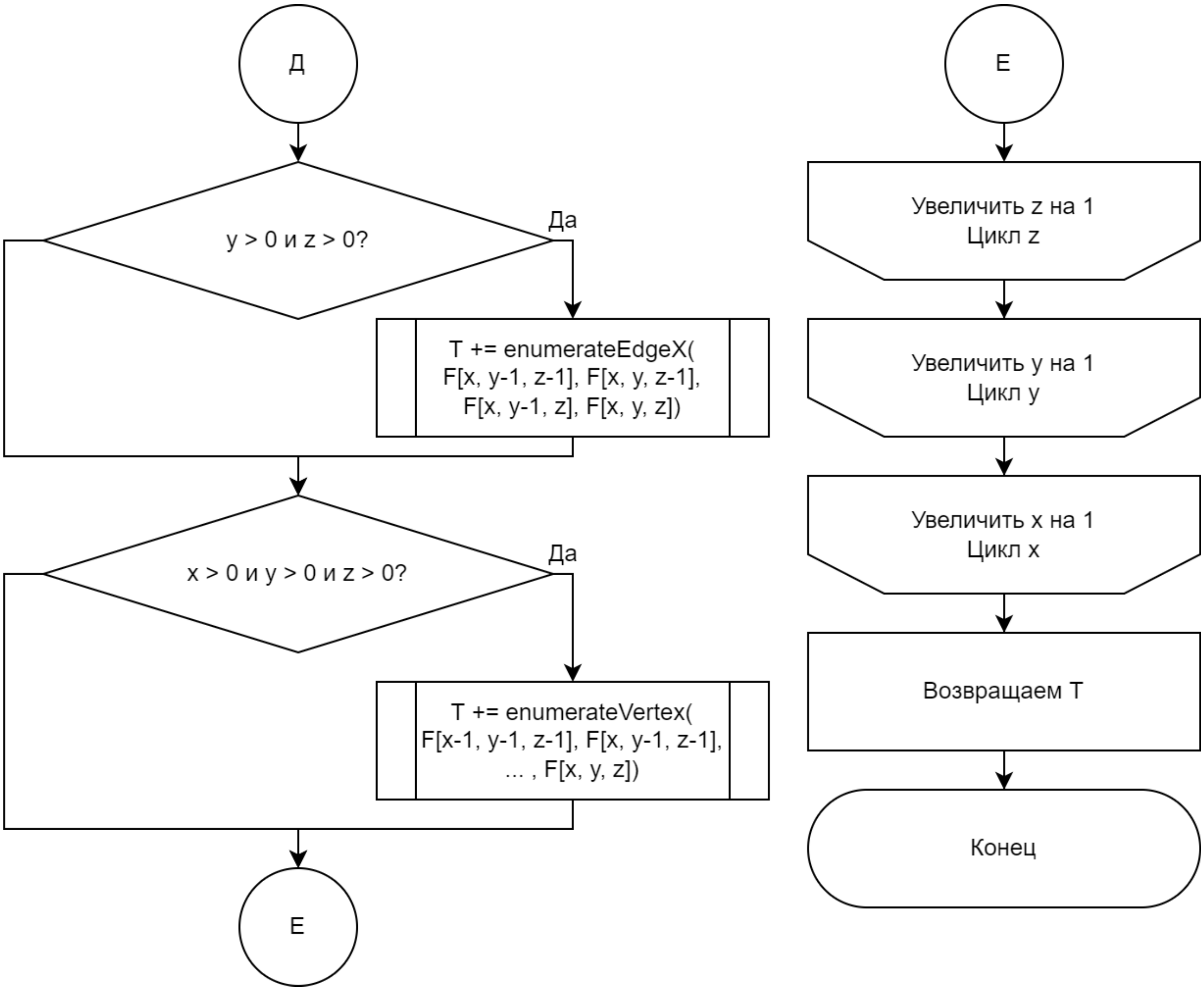


Рисунок 2.8 – Продолжение схемы функции `enumerateTriangles`

2.2 Описание используемых структур данных и оценка используемой памяти

В работе алгоритма используются следующие структуры данных:

- трехмерный вектор - 3 действительных числа. Занимает 24 байта;
- непрерывная функция - получается путем линейной интерполяции значений в слоях линейной томографии. Занимает (N, M, K) действительных чисел, где (N, M) - размер одного слоя; K - количество слоев. Так же имеет 2 трехмерных вектора описывающих ограничение области определения функции. Занимает $N * M * K * 8 + 48$ байт;
- треугольник - 3 вершины, описанные трехмерными векторами. Занимает 72 байта памяти;
- восьмеричное дерево - древовидная структура, узлами которой может быть или поддерево с 8-ю потомками, или лист с координатами вершины и значением функции в ней. Узел-поддерево занимает 64 байта, лист - 32 байта.

2.3 Структура разрабатываемого программного обеспечения

На рисунке 2.9 представлена UML диаграмма разрабатываемого программного обеспечения.

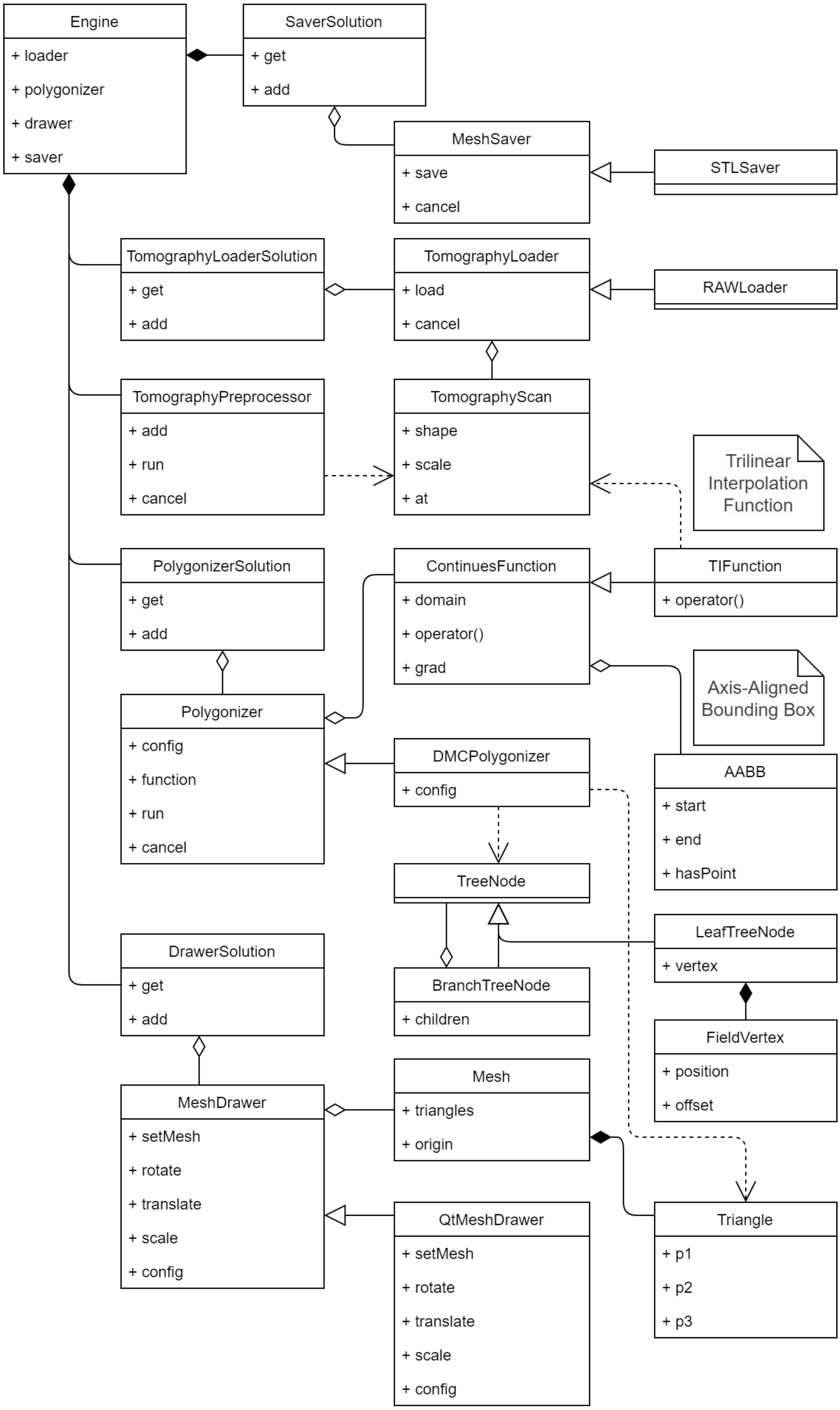


Рисунок 2.9 – UML диаграмма разрабатываемого программного обеспечения

Вывод

В данном разделе были приведена схема алгоритма Dual Marching Cubes, описаны используемые структуры данных, оценен объем памяти, необходимый для хранения данных, приведена UML диаграмма разрабатываемого приложения.

3 Технологический раздел

В данном разделе будут определены средства программной реализации, описан процесс сборки, пользовательский интерфейс и приведено функциональное тестирование.

3.1 Выбор средства программной реализации

Для реализации был выбран фреймворк Qt[5] на C++, так как он содержит базовые компоненты пользовательского интерфейса, что позволит больше сконцентрироваться на программировании основного алгоритма. Также для Qt существует интегрированная среда разработки QtCreator[6], упрощающая создание пользовательского интерфейса и отладку приложения.

Были использованы следующие библиотеки:

- **fmt**[7] - библиотека форматирования текста. Используется в генерации сообщений ошибок;
- **Eigen**[8] - библиотека для линейной алгебры. Используется только метод Якоби для решения системы линейных уравнений.

3.2 Процесс сборки приложения

Для сборки приложения из исходников понадобится Qt4.x[9] и QtCreator. В QtCreator нужно открыть папку проекта, выбрать сборку Release, собрать проект.

3.3 Пользовательский интерфейс

Пользовательский интерфейс состоит из области отображения полигональной модели и меню, разделенного по вкладкам. Каждая вкладка отражает один из этапов: загрузка томографии, преобразование в полигональную модель, настройка освещения и отображения модели, трансформация модели.

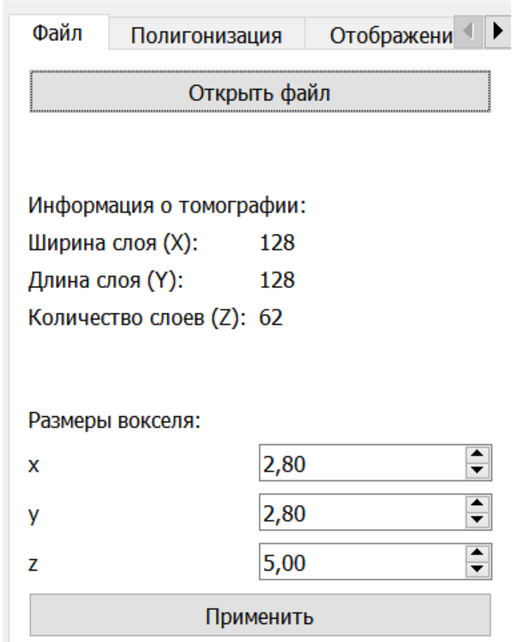


Рисунок 3.1 – Вкладка "Файл". На ней можно загрузить файл томографии, просмотреть информацию о загруженном файле, отредактировать размеры вокселя

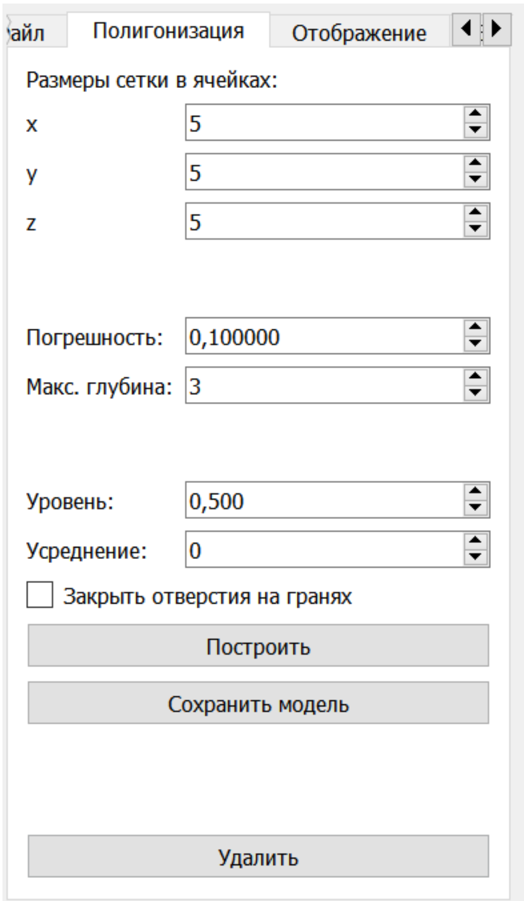


Рисунок 3.2 – Вкладка "Полигонизация". На ней представлены параметры алгоритма полигонизации, кнопки сохранения и удаления созданной модели

На рисунке 3.2 видны дополнительные параметр *Усреднение* и флаг *Закрыть отверстия на гранях*.

В поле *Усреднение* указывается расстояние до соседних вокселей, среди которых вычисляется среднее значение. Это позволяет получать более сглаженные модели на зашумленных данных томографии.

Установленный флаг *Закрыть отверстия на гранях* добавляет по одному пустому вокселю ко всем граням томографии, для закрытия отверстий, образовавшихся на гранях. Они появляются, потому что объект не поместился целиком в область томографии.

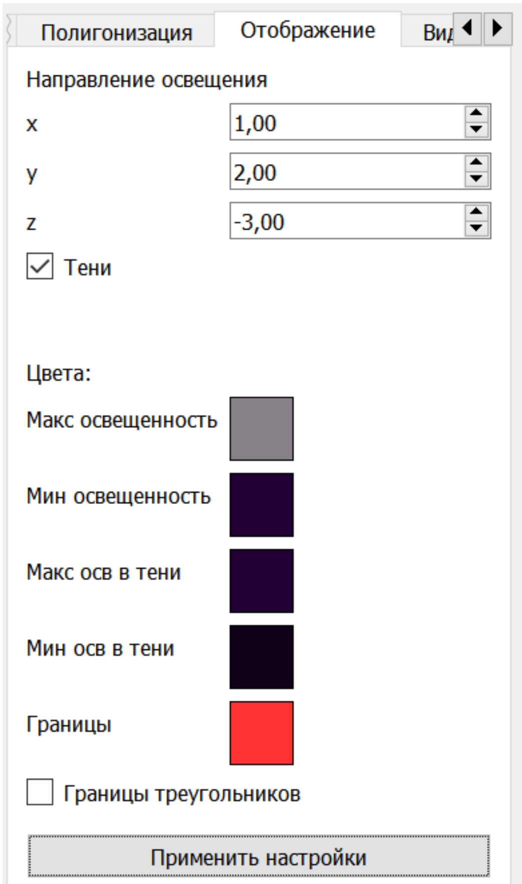


Рисунок 3.3 – Вкладка "Отображение". В ней можно настроить освещение и тени

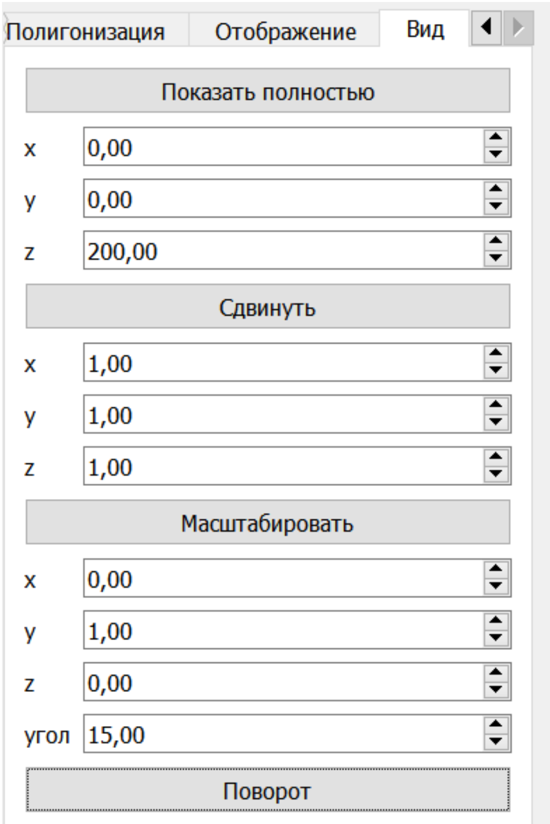


Рисунок 3.4 – Вкладка "Вид". В ней можно применить простые трансформации к модели для просмотра результата

3.4 Форматы входных и выходных данных

Для хранения томографии был выбран формат файлов проекта OpenQVis[10] из-за своей простоты чтения и записи. Томография хранится в двух файлах: заголовочном (`.dat`) и файлом вокселей(`.raw`).

Для хранения полигональной модели был выбран формат файла STL(бинарная версия)[11], так как является простым и широко поддерживаемым форматом.

3.5 Пример работы приложения

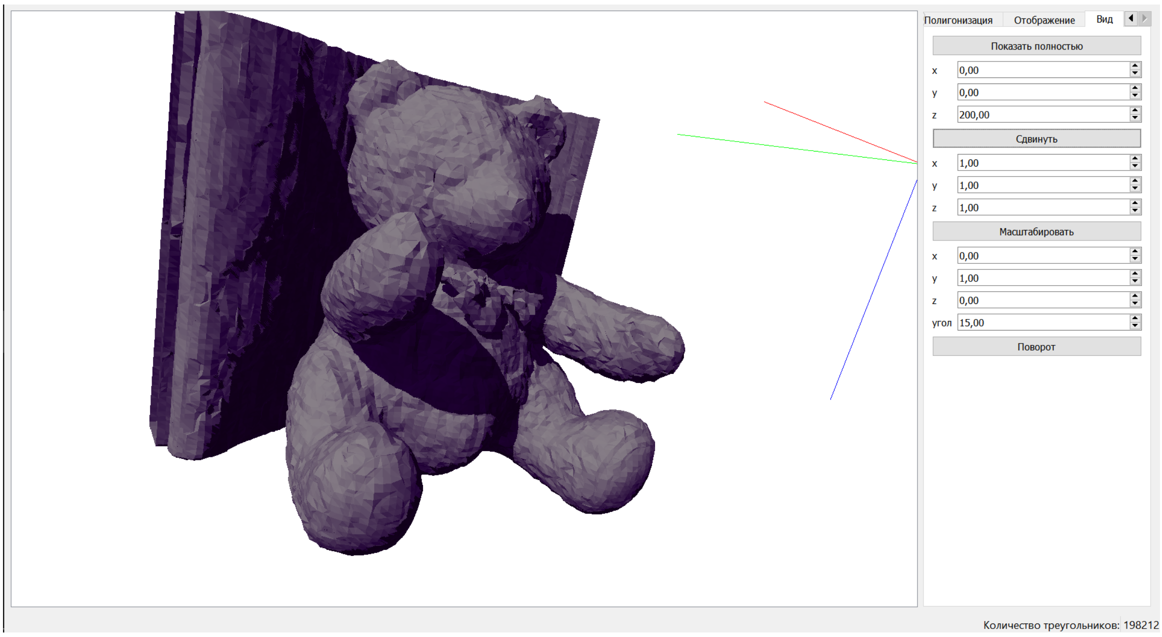


Рисунок 3.5 – Пример результата работы приложения. Полигонизация томографии плюшевого мишки

3.6 Тестирование программного обеспечения

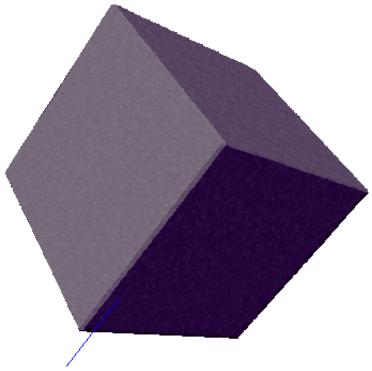
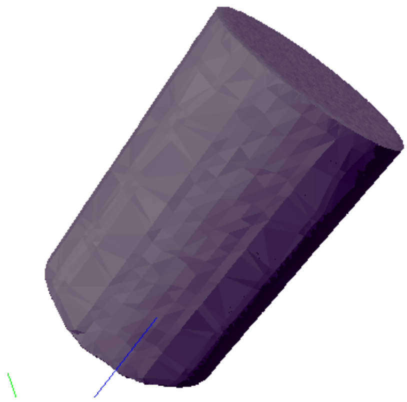
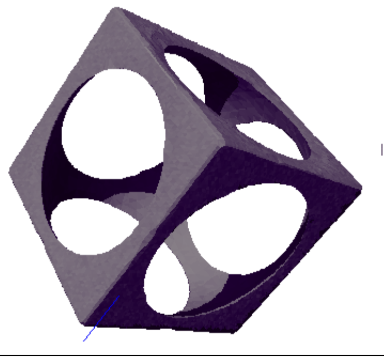
Функциональное тестирование проведено с помощью известных геометрических объектов. Для этого программно были созданы их томографии. Размер томографии - (128, 128, 128). В таблице 3.1 представлены функции, описывающие объект, и результат полигонизации. Причем, функция s - функция размывания границы, принимает знаковое расстояние до поверхности и коэффициент размытия, имеет вид:

$$s(v, w) = \begin{cases} 1 & \text{если } v < -w \\ \frac{w-v}{2w} & \text{если } -w \leq v < w \\ 0 & \text{если } v \geq w \end{cases} \tag{3.1}$$

Вывод

В данном разделе были определены средства программной реализации, описан процесс сборки приложения, пользовательский интерфейс и приведено функциональное тестирование.

Таблица 3.1 – Функциональные тесты

Название	Функция f	Результат
Шар с радиусом 40	$s(\sqrt{x^2+y^2+z^2}-40,5)$	
Куб со стороной 80	$s(\max(\lvert x\rvert,\lvert y\rvert,\lvert z\rvert)-$ $-40,0)$	
Цилиндр с радиусом основания 40 выстой 120	$s(\sqrt{x^2+y^2}-40,5)\cdot$ $\cdot s(\lvert z\rvert-60,0)$	
Куб со стороной 80, в котором вырезали шар радиусом 50	$s(\max(\lvert x\rvert,\lvert y\rvert,\lvert z\rvert)-$ $-40,0)\cdot(1-s($ $\sqrt{x^2+y^2+z^2}-50,5))$	

4 Исследовательский раздел

В данном разделе будет поставлен эксперимент по оценки эффективности работы алгоритма с использованием параллельности.

4.1 Цель эксперимента

Рассматривая схему алгоритма Dual Marching Cubes 2.1-2.8, можно заметить, что построение отдельных восьмеричных деревьев, как и нахождение треугольников изоповерхности не зависит от их соседей. Таким образом, возможно распараллеливание вложенных циклов на рисунках 2.2 и 2.6.

Целью эксперимента является оценка временной эффективности параллельной реализации алгоритма Dual Marching Cubes.

4.2 Описание эксперимента

С помощью библиотеки `OpenMP`[12] было реализовано параллельное выполнение циклов на рисунках 2.2 и 2.6.

На 1 этапе алгоритма создаётся много узлов восьмеричного дерева, что вызывает блокировки потоков вовремя выделения памяти. Чтобы их обойти был создан специальный класс `ObjectPool`, который выделяет память заранее и при необходимости удваивает ее.

Эксперимент проводился на компьютере со следующими характеристиками.

- Операционная система: Windows;
- Память: 16 GiB;
- Процессор: Intel i5-10210U CPU @ 1.60GHz;
- Количество логических потоков: 8.

В таблице 4.1 представлены результаты эксперимента: усредненное за 40 прогонов время полигонизации томографии мишки в зависимости от размеров сетки и максимальной глубины восьмеричных деревьев.

Таблица 4.1 – Среднее время полигонизации параллельной и последовательной реализации алгоритма.

Макс. глубина деревьев, ед.	**Размер сетки, ячеек**	**Время, мс**	
		Послед. реал.	**Паралл. реал.**
1	1	7	12
1	5	897	234
1	10	6382	1566
1	15	17894	4372
1	20	37670	12167
2	1	66	108
2	5	6677	2495
2	10	37609	13531
2	15	101519	36492
2	20	196905	71413
3	1	548	714
3	5	39680	14813
3	10	197618	72409
3	15	448713	163171
3	20	-	258929

Вывод

В данном разделе был проведен эксперимент по параллелизации алгоритма Dual Marching Cubes.

В результате эксперимента были получены следующие результаты:

- последовательная реализация алгоритма имеет меньшее время для любой максимальной глубины деревьев только при размере сетки в 1 ячейку (при макс. глубине 3 последовательная реализация быстрее на 23%). Это связано с потерями на создание отдельных потоков, инициализацию и использования дополнительных структур;

- параллельная реализация быстрее последовательной при размерах сетки, больших 1 ячейки. Так при макс. глубине деревьев 1 и размере сетки в 5 ячеек параллельная реализация быстрее в 3.78 раз, а при макс. глубине 2 и размере сетки в 20 ячеек – быстрее в 2.75 раз;

- с увеличением максимальной глубины деревьев эффективность параллелизации падает, что согласовывается с законом Амдала: вместе с увеличением максимальной глубины восьмеричного дерева, увеличивается и время построения самого глубокого дерева, то есть увеличивается время самого медленного последовательного фрагмента задачи. При размере сетки в 10 ячеек и макс. глубине 1 параллельная реализация быстрее в 4.07 раз, при макс. глубине 2 – в 2.78 раз, а при глубине 3 – в 2.73 раза.

ЗАКЛЮЧЕНИЕ

Во время выполнения курсовой работы было сделано следующее:

- изучены и проанализированы алгоритмы компьютерной графики построения полигональных моделей из послойных снимков;
- подробно изучен Dual Marching Cubes для применения к поставленной задаче;
- была приведена схема алгоритма Dual Marching Cubes;
- описаны используемые структуры данных;
- описана структура разрабатываемого программного обеспечения с помощью диаграммы UML;
- определены средства программной реализации;
- разработано и протестировано программное обеспечение;
- проведено исследование по параллелизации алгоритма Dual Marching Cubes.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *W. E. Lorensen H. E. C.* Marching cubes: A highresolution 3d surface construction algorithm. In Computer Graphics (Proceedings of SIGGRAPH 87). — 1987.

2. *Wehle M.*, *MГjller H.* Visualization of Implicit Surfaces Using Adaptive Tetrahedrizations // Dagstuhl '97 - Scientific Visualization Conference. — Los Alamitos, CA, USA : IEEE Computer Society, 06.1997. — С. 243. — DOI: `10.1109/DAGSTUHL.1997.10005`. — URL: `https://doi.ieeecomputersociety.org/10.1109/DAGSTUHL.1997.10005`.

3. T. Ju, F. Losasso, S. Schaefer, J. Warren. Dual Contouring of Hermite Data [Электронный ресурс]. — Режим доступа: `https://www.cs.rice.edu/~jwarren/papers/dualcontour.pdf` дата обращения: 16.11.2021).

4. S. Schaefer, J. Warren. Dual Marching Cubes: Primal Contouring of Dual Grids [Электронный ресурс]. — Режим доступа: `https://www.cs.rice.edu/~jwarren/papers/dmc.pdf` дата обращения: 16.11.2021).

5. Qt | Cross-platform software development for embedded desktop [Электронный ресурс]. — Режим доступа: `https://www.qt.io` дата обращения: 16.11.2021.

6. qt-creator/qt-creator: A cross-platform Qt IDE [Электронный ресурс]. — Режим доступа: `https://github.com/qt-creator/qt-creator` дата обращения: 16.11.2021.

7. Overview — fmt 8.0.1 documentation [Электронный ресурс]. — Режим доступа: `https://fmt.dev` дата обращения: 16.11.2021.

8. Eigen [Электронный ресурс]. — Режим доступа: `http://eigen.tuxfamily.org` дата обращения: 16.11.2021.

9. Index of /archive/qt/4.8/4.8.4 [Электронный ресурс]. — Режим доступа: `https://download.qt.io/archive/qt/4.8/4.8.4/` дата обращения: 16.11.2021.

10. The OpenQVis Project at sourceforge.net [Электронный ресурс]. — Режим доступа: `http://openqvis.sourceforge.net/docu/fileformat.html` дата обращения: 16.11.2021.

11. The StL Format | fabbers.com [Электронный ресурс]. — Режим доступа: http://www.fabbers.com/tech/STL_Format дата обращения: 16.11.2021.
12. Home - OpenMP [Электронный ресурс]. — Режим доступа: https://www.openmp.org/ дата обращения: 16.11.2021.