МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»

Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой ЭВМ
_____ Д.В. Топольский
«___»_____ 2024 г.

Разработка кроссплатформенной графической библиотеки на основе воксельного метода для визуализации объектов различной типологии

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

Руководитель работы,
к.т.н., доцент каф. ЭВМ
_____Ю.Б. Кухта
«___»_____2024 г.

Автор работы,
студент группы КЭ-406
_____И.В. Ефимов
«___»_____2024 г.

Нормоконтролёр,
ст. преп. каф. ЭВМ
_____ С.В. Сяськов
«___»_____2024 г.

Челябинск-2024

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

УТВЕРЖДАЮ
Заведующий кафедрой ЭВМ
_____Д.В. Топольский
«___»_____2023 г.

ЗАДАНИЕ

на выпускную квалификационную работу бакалавра

студенту группы КЭ-406
Ефимову Илье Вячеславовичу
обучающемуся по направлению
09.03.01 «Информатика и вычислительная техника»

1. **Тема работы: «**Разработка кроссплатформенной графической библиотеки на основе воксельного метода для визуализации объектов различной типологии»

2. **Срок сдачи студентом законченной работы:** 01 июня 2024 г.

3. **Исходные данные к работе:**

3.1. Минимальные аппаратные требования:

- Процессор Intel Pentium 4;
- Объем ОЗУ – 256 мегабайт.

3.2. Операционные системы:

- Windows XP и выше;
- Linux.

3.3. Положение камеры: декартовые координаты в вещественных единицах.

3.4. Ориентация камеры: радианные значения углов Эйлера.

3.5. Тестовые трехмерные модели.

- Модель «Стенфордский кролик»;
- Модель «Гермес, завязывающий сандалию»;
- Модель «Атриум дворца Спонца».

4. **Перечень подлежащих разработке вопросов:**

Выпускная квалификационная работа (ВКР) должна содержать разработку следующих вопросов:

4.1. Аналитический обзор научно-технической, нормативной и методической литературы по тематике работы.

4.2. Разработка алгоритмов и структур данных, позволяющих осуществить рендеринг воксельных моделей.

4.3. Реализация разработанных алгоритмов и структур данных.

4.4. Проведение тестирования реализованных алгоритмов.

Дата выдачи задания: 1 декабря 2023 г.

Руководитель работы _____/ *Ю.Б. Кухта/*

Студент _____/ *И.В. Ефимов /*

КАЛЕНДАРНЫЙ ПЛАН

Этап	Срок сдачи	Подпись руководителя
Аналитический обзор научно-технической, нормативной и методической литературы по тематике работы	01.03.2024	
Разработка алгоритмов и структур данных, позволяющих осуществить рендеринг воксельных моделей	01.04.2024	
Реализация разработанных алгоритмов и структур данных	01.05.2024	
Проведение тестирования реализованных алгоритмов	15.05.2024	
Компоновка текста работы и сдача на нормоконтроль	24.05.2024	
Подготовка презентации и доклада	30.05.2024	

Руководитель работы _____/ *Ю.Б. Кухта* /

Студент _____/ *И.В. Ефимов* /

АННОТАЦИЯ

Ефимов И.В Разработка кроссплатформенной графической библиотеки на основе воксельного метода для визуализации объектов различной типологии – Челябинск: ФГАОУ ВО «ЮУрГУ (НИУ)», ВШ ЭКН; 2024, 52 с., библиогр. список – 20 наим.

В рамках выпускной квалификационной работы проведен аналитический обзор современной научно-технической, нормативной, методической литературы, затрагивающей научно-техническую проблему.

Разработаны и реализованы алгоритмы и структуры данных, позволяющие получать двухмерные растровые изображения на основе трехмерных воксельных моделей.

С использованием разработанной библиотеки создана тестовая программа, в результате выполнения которой была проведена проверка функционала созданной графической библиотеки, по итогам которой получены корректные двухмерные изображения трехмерных воксельных моделей.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ 7
1. АНАЛИТИЧЕСКИЙ ОБЗОР НАУЧНО-ТЕХНИЧЕСКОЙ, НОРМАТИВНОЙ И МЕТОДИЧЕСКОЙ ЛИТЕРАТУРЫ ПО ТЕМАТИКЕ РАБОТЫ 9
1.1. Воксельный подход для визуализации трехмерных изображений 9
1.2. Применение воксельного метода в теории и на практике 12
1.4. Актуальность проблемы 24
2. РАЗРАБОТКА АЛГОРИТМОВ И СТРУКТУР ДАННЫХ, ПОЗВОЛЯЮЩИХ ОСУЩЕСТВИТЬ РЕНДЕРИНГ ВОКСЕЛЬНЫХ МОДЕЛЕЙ 26
2.1. Функциональные требования 26
2.2. Нефункциональные требования 26
2.3. Структура библиотеки 27
2.4. Достижение кроссплатформенности. 28
3. РЕАЛИЗАЦИЯ РАЗРАБОТАННЫХ АЛГОРИТМОВ И СТРУКТУР ДАННЫХ 29
4. ПРОВЕДЕНИЕ ТЕСТИРОВАНИЯ РЕАЛИЗОВАННЫХ АЛГОРИТМОВ .. 43
ЗАКЛЮЧЕНИЕ 50
БИБЛИОГРАФИЧЕСКИЙ СПИСОК 51

ВВЕДЕНИЕ

Компьютерная графика – совокупность методов создания и редактирования изображений с помощью ЭВМ и специального программного обеспечения. В настоящее время сложно найти область науки и промышленности, где бы не использовались методы визуализации объектов, компьютерное зрение либо алгоритмы распознавания образов. В настоящее время результаты исследований в области компьютерной графики используются в следующих отраслях человеческой деятельности:

1) Наглядная визуализация научных расчетов и данных экспериментов.
2) Создание деловых иллюстраций, таблиц, графиков, диаграмм и прочих средств для представления деятельности разных учреждений.
3) Создание чертежей в САПР с возможностью получения 3D изображения и проекций деталей.
4) Создание художественных иллюстраций, плакатов, видеороликов, компьютерных игр, презентаций.
5) Создание движущихся изображений – анимаций.
6) Создание моделей для 3D-печати.
7) Цифровая живопись.

Компьютерная графика делится на двухмерную – построение изображений на плоскости, и трехмерную – построение изображений в пространстве. Также графика делится на растровую и векторную. Векторная графика – построение изображений с помощью математического описания изображаемых объектов. Растровая графика – построение изображений из элементарных частиц. В двухмерной графике ими являются пиксели, в трехмерной – воксели. Одной из актуальных задач компьютерной графики является повышение качества получаемых изображений. Для этого используют следующие методы:

1) Увеличение числа одновременно изображаемых элементов.
2) Совершенствование математических моделей освещения.
3) Сглаживание.
4) Фильтрация.
5) Применение нейронных сетей.

Методы для повышения качества получаемого изображения или скорости его генерации постоянно совершенствуются, так как точность и детализация с максимальной визуализацией изображения очень важна для реализации многих графических задач, например, распознавание реальных объектов при помощи

фото и видео ряда. Большинство современных технологий и инструментов в этой области разработано для полигонального представления трехмерных объектов, то есть с помощью набора минимальных поверхностей. Это связано с тем, что на заре компьютерной графики, алгоритмы растеризации полигонов могли выполняться с приемлемой скоростью на ЭВМ того времени, тогда как альтернативы требовали больших вычислительных мощностей. Это привело к тому, что платы видеоускорения с тех пор создавались с расчетом, в первую очередь, на ускорение с помощью параллельного распределения на множестве вычислительных ядер процесса решения задачи растеризации треугольников, то есть превращения заданного тремя вершинами векторного представления треугольника в растровое изображение, которое можно вывести на пиксельный экран. Поэтому организация отдельных производственных мощностей для обеспечения аппаратной поддержки альтернатив полигонам является очень дорогостоящей инвестицией на настоящий момент [1].

Тем не менее, отсутствие аппаратной поддержки стимулировало развитие алгоритмической части альтернатив полигональному методов решения задачи визуализации изображений, например – воксельный рендеринг.

Воксельное представление трехмерных моделей является логическим следствием развития современных методов улучшения качества изображений, получаемых с помощью трехмерных полигональных моделей, а именно – увеличения количества одновременно изображаемых на экране полигонов. Большое количество полигонов достигается за счет уменьшения размера каждого отдельного полигона, что постепенно приводит к уменьшению полигона до размеров одного пикселя, следовательно, многоступенчатый процесс растеризации становится излишним, так как более быстрым решением является непосредственная установка цвета этого пикселя, для чего могут использоваться более оптически достоверные методы получения изображения, такие как трассировка или бросание лучей, трассировка путей, метод фотонных карт и другие, применение которых оправдано со структурами данных, приближенных к устройству объектов реального мира. Помимо этого, воксельное представление трехмерных моделей позволяет описывать не только внешнюю поверхность объекта, но также и его внутреннюю структуру.

В силу вышеуказанных причин, на сегодняшний день актуальной является разработка кроссплатформенного инструмента для альтернативного подхода к решению задачи визуализации – с помощью воксельного метода.

1. АНАЛИТИЧЕСКИЙ ОБЗОР НАУЧНО-ТЕХНИЧЕСКОЙ, НОРМАТИВНОЙ И МЕТОДИЧЕСКОЙ ЛИТЕРАТУРЫ ПО ТЕМАТИКЕ РАБОТЫ

1.1. Воксельный подход для визуализации трехмерных изображений

Воксельный подход для визуализации трехмерных изображений известен достаточно продолжительное время, появившись в 1960-е годы, он стал объектом исследований возможностей его применения в среде систем автоматизированного проектирования [2], первые статьи на эту тему появились в 1980-х годах, где термин «воксель» определяется как непрерывное пространство, удовлетворяющее следующему неравенству (1.1):

$$V_i - \frac{\delta}{2} \leq x_i < V_i + \frac{\delta}{2}, (i = 1, 2, 3), \quad (1.1)$$

где V – элемент регулярно распределенного массива точек S, ед.;

δ – единичная длина, чьи кратные величины являются координатами S, ед.;

x – координаты рассматриваемого непрерывного пространства, ед.

Термин «воксель» это сокращение от «volume element», по аналогии с «пиксель» - «picture element» [3].

Существует два основных семейства способов рендеринга воксельных моделей: извлечение изоповерхностей (полигональные способы), и вариации трассировки лучей.

Способы извлечения изоповерхностей, или построение полигональных моделей из воксельных данных:

1) Алгоритм «Marching cubes» [4]

Суть алгоритма заключается в постепенном обходе всего воксельного пространства, в процессе которого рассматриваются близлежащие ячейки. В зависимости от их состояния, выбирается одна из 256 возможных конфигураций полигонов в рассматриваемой ячейке.

Из достоинств алгоритма можно выделить легкое распараллеливание, а также возможность значительного ускорения работы, благодаря возможности предварительного расчета всех 256 конфигураций ячеек.

Существенным недостатком алгоритма является невозможность создания поверхностей расположенных под прямым углом друг к другу. Также создаваемая полигональная сетка может получиться немногообразной.

2) Алгоритм «Extended marching cubes» [5]

Алгоритм «Marching cubes» справляется со своей задачей в большинстве случаев, однако у него оставались проблемы, например, в некоторых случаях генерировались вырожденные треугольники, или треугольники с очень маленькими площадями, сторонами или углами. Для решения этих проблем было разработано дополнение к оригинальному алгоритму.

Изначальная справочная таблица, содержащая все возможные конфигурации полигонов, и строящаяся по принципу двух меток для каждой вершины («больше» или «меньше»), получает расширение, и ее итоговый размер становится 3^8 элементов, так как добавляется третий тип меток – «равно». Расширение таблицы позволило избавится от постобработки, которая использовалась в оригинальном алгоритме для исправления вырожденных треугольников. Сам ход алгоритма при этом получил серьезных изменений, все также осуществляется проход от одного вокселя к другому, с выбором подходящей конфигурации из таблицы. Несмотря на это, расширенная версия алгоритма все еще имеет свои недостатки: близко расположенные, но не соединенные элементы могут оказаться соединенными, а также существуют возможности построения немногообразных полигональных сеток.

3) Алгоритм «Surface nets» [6]

Этот алгоритм является одним из первых в группе «двойственных» алгоритмов. Двойственность заключается в том, что получаемая модель образует двойственный граф. Работа алгоритма это проход через все ячейки пространства, но вместо выбора из возможных конфигураций, положение каждой вершины рассчитывается как среднее между соседними вокселями.

Достоинство метода заключается в простоте реализации, и меньшим количеством полигонов в итоговой модели, по сравнению с алгоритмом «Marching cubes».

Недостатки – как и все двойственные методы, «Surface nets» трудно распараллелить, а также существует возможность получения немногообразной сетки и отсутствие возможности получения прямых углов.

4) Алгоритм «Dual contouring of Hermite data» [7]

Чтобы избавится от явной обработки особых случаев, таких как создание прямых углов, был разработан новый алгоритм, упрощающий процесс контурирования – то есть создание полигональной сетки через соединение сгенерированных вершин в не пустых ячейках воксельного пространства.

В качестве требования к исходным «эрмитовым» данным, алгоритму необходимо представить воксельное пространство как знаковое скалярное поле, элементы которого интерпретируются как пересечения модели с границами вокселей и определенные нормали к этим вершинам.

Метод является прямым развитием идеи, представленной в алгоритме «Extended marching cubes», а именно использование функции квадратичной ошибки для определения положения каждой вершины (1.2):

$$E[x] = \sum_i (n_i * (x - p_i))^2, \quad (1.2)$$

где p – координаты точки пересечения с границами вокселя, ед;

n – нормаль к этой точке, ед.

Итоговая вершина ставится в минимуме вышеуказанной функции, для каждого вокселя в котором происходит смена знака. Дополнительно, такой подход позволяет избежать постобработки «заделывания трещин» в получившейся модели.

Из недостатков – алгоритм требует комплексную структуру исходных данных, а также создает немногообразные сетки.

5) Алгоритм «Adaptive skeleton climbing» [8]

Алгоритмы реализующие кубический подход к извлечению изоповерхностей характеризуются большим количеством созданных полигонов в конечной модели, что является проблемой при использовании в системах где необходима визуализация в реальном времени.

Данный алгоритм принципиально отличен от кубических методов, он способен выполняться со сравнимой с ними скоростью, при этом количество полигонов в результате, в зависимости от сложности исходных данных, меньше в диапазоне от 4 до 25 раз, не требуя постобработки.

Ход работы алгоритма заключается в постепенном извлечении изоповерхностей на разных разрешениях – сначала находятся изовершины на границах воксельной сетки, затем происходит построение изолиний на гранях, и наконец – изоповерхностей в кубах. Отличительной особенностью алгоритма также является адаптивность – размер рассматриваемых кубов адаптируется к структуре модели, то есть там где необходимо более гладкое представление поверхности используются более большие объемы кубов. В силу постепенности алгоритма (переход от грубого представления к точному), данный метод имеет еще пару преимуществ перед операцией упрощения полигональной сетки в процессе дополнительной постобработки – более грубые участки модели создаются быстрее, а также становится возможно получать промежуточные результаты работы алгоритма на лету, что позволяет дать пользователю принимать решение о необходимости последующей более детальной обработки.

1.2. Применение воксельного метода в теории и на практике

В последнее время применение воксельной технологии широко распространено в связи развитием полигональных технологий. Представим примеры применения.

Воксельная сетка используется для ускорения процесса трассировки лучей. Разбив пространство на ячейки, становится возможно не проверять пересечение лучей со всеми объектами сцены, а, проходя от одного вокселя к другому, вычислять конкретный объект с которым пересекается луч [9].

Алгоритмы вокселизации это алгоритмы для переноса полигональных моделей в воксельное пространство. Это достигается созданием воксельного пространства вокруг модели, после чего совершается постепенный проход через все ячейки, в процессе которого определяется пересекает модель воксель или нет. Вокселизации как предварительной обработке подвергаются модели для использования в последующем эффективных алгоритмов расчета эффектов теней и отражений [10].

Классические алгоритмы расчета теней с помощью определения траектории светового луча вычислительно дорогостоящи. Воксели позволяют применять удобные приближения, дающие мало отличимые от классических алгоритмов результаты, но по гораздо меньшему вычислительному объему. Например, в одном из предлагаемых методов, воксельное пространство послойно, с итеративно увеличивающейся силой сдвига, смещается в противоположную от источника освещения сторону, после чего участки, оказывающиеся закрытыми сверху другими вокселями, затеняются [11].

Вокселизация экрана в разных разрешениях используется для быстрого определения видимых и невидимых мест. Это достигается дискретизацией пространства, попавшего в пирамиду видимости, представляя каждый набор компонентов цвета каждого пикселя как координаты вокселя. Получившаяся структура создается за один проход отрисовки, и потому может использоваться в приложениях реального времени для создания эффектов рефракции и полупрозрачности. Также, эта структура может использоваться как перспективная сетка, в качестве простой ускоряющей структуры для трассировки лучей [12].

В медицине, для создания моделей внутренних органов в прошлом использовался математический подход, который заключался в описании форм органов с помощью геометрических примитивов (плоскостей, сфер, цилиндров, конусов, эллипсоидов, эллиптических цилиндров, фрагментов всех этих фигур и их комбинаций), заданных уравнениями. Тело человека в таких моделях, названных MIRD (в честь Медицинского Комитета по Внутренним Радиационным Дозам) представлялось как эллиптический цилиндр, к которому добавлялись аналогичные замкнутые области пространства, представляемые как голова, шея, руки и ноги. Такие приближения оказались неэффективными, так как они слабо походили на настоящие формы внутренних органов, математически заданные органы плохо соединялись друг с другом, а необходимость уместить

органы в эллиптическое туловище приводило к тому, что получаемое расположение органов имело мало общего с анатомической действительностью. Развитие методов компьютерной томографии и магнитно-резонансной томографии позволило представлять внутренние органы как воксельные модели, которые в точности описывали форму и состав органов, помечая каждый воксель как принадлежащий тому или иному органу или ткани [13].

Развитие технологий производства камер глубины, вместо цвета собирающих информацию о расстоянии до объектов, и увеличение доступности таких устройств широким слоям населения (например, Microsoft Kinect или Asus Xtion), привело к необходимости развития программных средств для обработки получаемых с этих устройств данных. Технологии на основе облаков точек просты по структуре и близки к тому, с чем работают сами сенсоры камер глубины, но основной недостаток заключается в отсутствии средств для эффективной перестройки в реальном времени получаемых поверхностей. Представление с помощью карт высот позволяет добиться хороших результатов в сжатии информации, и способно легко масштабироваться для больших сцен, но неспособно реконструировать комплексные трехмерные структуры. Решением стало использование хешируемых объемных структур, совмещающих как достоинства обычных воксельных сеток, но при этом экономно занимающих память, что позволяет в реальном времени объединять получаемые части информации о сканируемых поверхностях, визуализировать их, а также способных быстро добавлять или удалять информацию, что позволяет работать с перемещающейся камерой глубин [14].

В не имеющей отношения к компьютерной графике области науки – пространственном анализе, воксели также находят свое применение. Так, вокселизация трехмерных полигональных моделей используется как перспективный метод для динамической генерации анимаций, которые в дальнейшем используется для автоматической презентации моделей или сцен. В данном случае, под вокселями понимается разбиение множества объектов в сцене на кубическую сетку, где каждая ячейка содержит информацию о субмоделях, лежащих в этой области пространства. Применение воксельной технологии позволяет абстрагироваться от комплексной природы исследуемых объектов при выполнении расчетов, а также позволяет работать с очень сложными системами, которые имеют свойство частого изменения или обладают большими размерами [15].

В геологии, для составления карт подземных структур, успешно используются методы аэроэлектромагнитного сканирования, однако конвертация получаемой информации об удельном сопротивлении в литологические данные не является прямолинейной задачей, так как необходимо учитывать множество ограничений при интерпретации получаемой информации как трехмерной геологической модели. Несмотря на сложности, получаемые модели остаются бесценными уже на протяжении 30 лет для исследования грунтовых вод, а также архивирования уже имеющейся информации. Ранее, для составления таких моделей,

применялись послойные модели, которые конечно имеют преимущество в виде простого построения при однородной структуре исследуемых участков, но имеют большие проблемы в обратном случае – когда зона имеет сложное и неоднородное строение, к тому же при большом количестве слоев сильно уменьшается детализация модели. Применение как регулярных так и нерегулярных воксельных сеток позволило преодолеть недостатки послойных моделей, так как каждому вокселю можно четко задать определенные геологические характеристики. И хотя воксельный метод также не лишен недостатков, таких как сложности в ручном создании моделей, и в определении идеальных границ слоев или полостей, но зато для получаемых сеток становится возможно применять хорошо известные методы трехмерной интерполяции для расчета свойств вокселей, на основе данных полученных из скважин или из геофизического анализа местности [16].

Способы навигации в закрытых пространствах гораздо сложнее чем на открытых, так как в большинстве случаев четких путей и направлений для ходьбы не существует, поскольку человек может перемещаться по всему пустому пространству. Обычно для вычисления кратчайшего пути создается предопределенная сетевая модель. Однако такие сети слишком абстрактны и не учитывают всего пространства, в котором могут находиться или перемещаться люди. Дополнительную сложность вносит то, что обычно используются двухмерные планы помещений, которые не учитывают все доступное для навигации пространство, а также не моделируют размер и форму лестниц. Для решения этих проблем в робототехнике и геоинформационных системах активно находят свое применение воксельные модели, которые можно быстро генерировать из облаков точек, и впоследствии применять для расчета траектории движения из любой точки пространства. К тому же, такие способы обладают чисто геометрической природой, и не требуют для применения технологий распознавания и классификации встречающихся на пути объектов [17].

1.3. Обзор аналогов ПО

На сегодняшний день в свободном доступе существует относительно небольшое количество воксельных рендереров, в силу того что их авторами обычно являются энтузиасты в данной области, которые не всегда доводят свои наработки до итоговой реализации или хотя бы до публикации информации о разработке.

1) PND3D – разработан Кеном Сильверманом, является развитием его более старого воксельного рендерера: «Voxlap», использует для визуализации алгоритм «Wave Surfing». [18]

2) Doommy – автор неизвестен, для визуализации применяется алгоритм «Ray Casting». [19]

3) Doon Engine – разработан Дэниэлом Элвеллом, используется в его игре для персональных компьютеров «Terra Toy», для рендеринга используется метод

«Path Tracing», что разительно увеличивает качество получаемых изображений. [20]

Таблица 1 – Обзор достоинств и недостатков аналогов ПО.

Рендерер	Достоинства	Недостатки
PND3D	1) Работа без видеоускорителя. Наличие платы видеоускорения - это серьезное требование к техническим спецификациям машины, на которой будет выполняться программа. На сегодняшний день стоимость актуальных, либо недавно бывших таковыми, плат видеоускорения составляет существенную долю от итоговой стоимости собираемой конфигурации персонального компьютера. Помимо этого, интерфейсы программирования приложений для GPU постоянно появляются, совершенствуются, меняются и устаревают. Одно из первых семейств плат видеоускорения от компании «3dfx Interactive» - Voodoo, использовало программный интерфейс Glide. Однако компания «3dfx Interactive» закрылась в 2001 году, новых видеокарт Voodoo не было выпущено, а Glide стал лишь частью истории	1) Разрешено использовать только в некоммерческих проектах. Несмотря на открытость кода, данный рендерер не является свободным, и его использование жестко ограниченно. Отсутствие возможности свободно модифицировать, распространять и продавать производные работы сильно ударило по популярности рендерера, так как у энтузиастов, усилиями которых происходит развитие данной области компьютерной графики в основном отсутствуют бюджеты или полноценные бизнес-планы, с которыми был бы смысл задумываться о коммерческом сотрудничестве с автором. В силу того, что одной из целей библиотеки это максимальная простота в распространении, она будет лицензирована на максимально свободных условиях. 2) Часть кода разработана на ассемблере. Язык ассемблера у каждой платформы свой, программа,

Продолжение таблицы 1

Рендерер	Достоинства	Недостатки
	компьютерной графики. На смену пришли видеокарты от компаний «Nvidia» и «AMD», а наиболее распространенными интерфейсами для разработки программ, использующих функционал графического ускорения стали «OpenGL» и «DirectX». Эти интерфейсы имеют наибольшее распространение в современном программном обеспечении, но прогресс не стоял на месте, и развитием OpenGL стал абсолютно новый промышленный стандарт для работы с платами видеоускорения – «Vulkan». Помимо этого, в 2014 компания «Apple» создала свой собственный интерфейс для работы на их же графических картах – «Metal». Так как воксели не имеют прямой поддержки на современных платах обработки графики, библиотека не должна зависеть от какого-либо промышленного интерфейса, чтобы в будущем, при необходимости, ее можно было быстро переписать для поддержки новой	имеющая ассемблерные вставки, будет намертво привязана к платформе для которой создана. Ассемблерные вставки несомненно полезны, если стоит задача максимальной оптимизации под конкретную платформу, однако, разрабатываемая библиотека ставит перед собой иную задачу – максимальные доступность и независимость от платформы, возможные оптимизации связанные с использованием конкретного функционала разных платформ будут являться опциональными, а не обязательными. 3) Использование октодеревьев. Октодерево – это разновидность древовидной структуры данных, отличающейся тем, что от каждого узла дерева отходит ровно 8 потомков. Октодеревья используются для эффективно разбиения трехмерного пространства, по сути являясь аналогом квадродеревьев, которые используются для разбиения двухмерной плоскости. Разреженным называется такое октодерево, в котором не все узлы содержат информацию об объекте, что

Продолжение таблицы 1

Рендерер	Достоинства	Недостатки
	аппаратуры. 2) Минимальные требования к CPU – Intel Pentium 4. Так как использование аппаратного ускорения ограничено, библиотека должна быть способной выполняться с приемлемой скоростью на наиболее слабых процессорах из возможных. Такой подход уменьшит размер технического долга библиотеки, то есть ей потребуется меньше серьезных доработок в будущем и увеличит аудиторию пользователей за счет людей с не самыми передовыми конфигурациями персональных компьютеров.	позволяет не проверять их в процессе трассировки лучей. Использование разреженных воксельных октодеревьев это достаточно популярный подход в области оптимизации процесса трассировки лучей. Октодерево действительно ускоряет проход луча по сцене, но это верно только для статичных сцен. Перестройка октодерева алгоритмически медленна и не интуитивно понятна. Помимо этого, октодерево требует больших затрат памяти для хранения всех своих частей. Будущая библиотека будет способна работать с динамическими сценами, а используемый формат сцены должен быть компромиссом между производительностью и простотой восприятия. 4) Гарантия стабильности работы только на объемах до 4096^3 вокселей. Ограничение на размер воксельных моделей не влияет на функциональность в большинстве случаев применения, так как предлагаемого объема обычно достаточно, тем не менее, воксели успешно используются в создании обширных и детализированных ландшафтов,

Продолжение таблицы 1

Рендерер	Достоинства	Недостатки
		для чего необходима поддержка пространств произвольного размера. Будущая библиотека должна содержать способ работы с постепенно подгружающимися пространствами.
Doommy	1) Минимальные требования к CPU – Intel Pentium 4. 2) Способность работать в многопоточном режиме. Трассировка лучей для каждого пикселя на экране способна выполняться независимо от остальных пикселей на экране, поэтому распараллеливание этого алгоритма является тривиальной задачей. Задействование методик параллельного программирования даст ощутимый рост в производительности. И хотя библиотека будет ориентироваться на выполнение в одном потоке, поддержка параллельного выполнения на центральном процессоре является одной из ключевых особенностей. 3) Использование метода LOD для ускорения алгоритма «бросания	1) Сильно зависит от устаревшей библиотеки SDL1. SDL1 (Simple DirectMedia Layer) хоть и является кроссплатформенной библиотекой, способной запускаться на множестве разных платформ, но это также означает что она не платформонезависима – ею нельзя воспользоваться, если разработчики явно не добавили поддержку каждой отдельной платформы. Помимо этого, первая версия этой библиотеки устарела очень давно – с выхода SDL2 в 2013 году, и в 2022 году началась разработка SDL3. Будущая библиотека не будет зависеть от SDL или подобной библиотеки для абстрагирования от системных вызовов, реализовать этот функционал под свою платформу программисту будет необходимо самостоятельно, но благодаря платформонезависимой архитектуре этот процесс

Продолжение таблицы 1

Рендерер	Достоинства	Недостатки
	лучей». Метод LOD (англ. Level Of Detail – уровень детализации) это способ оптимизации процесса рендеринга, заключающийся в использовании моделей разного качества, в зависимости от расстояния между объектом, использующим модель, и точкой обзора, то есть чем дальше объект от наблюдателя, и, соответственно, чем меньше места он занимает на экране, тем грубее становится его обработка. Трассировка лучей с точки зрения скорости выполнения является очень тяжелой операцией, поэтому в будущей библиотеке необходимо задействовать как можно большее количество оптимизаций, одновременно пытаясь и сохранить исходный код доступным для понимания, и не урезать основной функционал, например, накладывая строгие ограничения на обрабатываемые модели. 4) Производительность слабо зависит от размера и наполненности сцены.	должен быть тривиальным. 2) Жесткие требования к объему сцены: поддерживаются только размеры, кратные 32. Это ограничение является следствием особенностей реализации алгоритма LOD в конкретно этом рендерере. И хотя это ограничение не является фатальным, так как часто модели подгоняются под соответствующие размеры для удобства работы, но по возможности стоит избежать таких неочевидных требований. 3) Неизменяемое разрешение 640x480 пикселей. Фиксированное разрешение дает стабильность при организации архитектуры, но это не обеспечивает легкой портируемости и гибкой настройки. В будущей библиотеке должна быть поддержка динамического разрешения, это позволит обеспечить работу на любом доступном дисплее, а также даст возможность тонкой настройки производительности, так как скорость выполнения трассировки лучей напрямую зависит от разрешения, то есть если на целевой машине не будет хватать скорости рендеринга – пользователь

Продолжение таблицы 1

Рендерер	Достоинства	Недостатки
	Это достоинство является следствием использования представления пространства в виде регулярной сетки, это наиболее простой по структуре способ организации пространства, у которого есть как достоинства в виде быстрой модификации, быстрого доступа к элементам, независимость от содержания сцены, так и недостатки: как у любой регулярной структуры, регулярная сетка подвержена так называемой проблеме мяча на футбольном поле – найти значимое место среди превалирующего незанятого объема очень сложно. При планировании архитектуры библиотеки будет необходимо взвешено подойти к выбору способа организации пространства.	сможет плавно снижать качество, достигая приемлемого соотношения качества и производительности. 4) В многопоточном режиме использует только два потока центрального процессора. Это решение принято для простоты реализации, в многопоточном режиме экран разбивается пополам – по половине для каждого потока, и это дает ощутимый рост производительности. Тем не менее, разбиение на два потока это самая минимальная оптимизация, современные процессоры даже бюджетного сегмента обладают множеством ядер для параллельного выполнения программ, и в большинстве программ они простаивают без работы. В распараллеливаемой части библиотеки планируется реализация гибкой настройки количества используемых ядер, а также, по возможности, задействование вычислительных ядер карт видеоускорения. 5) Использование фиксированной палитры цветов. Использование фиксированной палитры

Продолжение таблицы 1

Рендерер	Достоинства	Недостатки
		позволяет уменьшить один из главных недостатков воксельного подхода для представления трехмерного пространства, а именно высокие затраты памяти. Реализация фиксированной палитры из 256 цветов позволяет интерпретировать каждый воксель всего лишь как индекс в массиве цветов, так каждый воксель в несжатом виде занимает всего лишь один байт. Будущая библиотека должна быть способна работать с фиксированной палитрой, либо иметь возможность по желанию пользователя конвертировать имеющийся набор данных в формат с фиксированной палитрой.
Doon Engine	1) Наиболее современен и активно разрабатываем. Поддержка это наиболее длительный и затратный этап жизненного пути программного продукта. Чтобы не зависеть от наличия бюджета на поддержку, либо от наличия желания поддерживать самостоятельно, библиотека будет разработана с расчетом на отсутствие нужды в особой процедуре поддержки, так как при	1) Только GPU реализация. 2) Потоковая передача данных вызывает артефакты при быстром обновлении отображаемых частей. Так как воксели занимают много памяти и стремительное обновление видимых частей требует скорейшей загрузки новой информации в видеопамять, в коммуникации между процессором и видеокартой образуется затор, когда разрядность шины

Продолжение таблицы 1

Рендерер	Достоинства	Недостатки
	свободном доступе к библиотеке, и доступной для восприятия архитектуре, любой желающий при наличии определенной необходимости самостоятельно модифицировать ее под возникшие нужды. 2) Поддерживает отражения, трассировку путей для повоксельного освещения. Трассировка путей это особый частный случай метода трассировки лучей, трассировка путей стремится к наиболее физически достоверной симуляции поведения света. Основная особенность такого подхода заключается в том, что луч света отражается от встречаемых поверхностей и постепенно теряет свою интенсивность, пока не поглотится, либо не рассеется. С трассировкой путей можно легко достичь эффектов, которые имеют очень сложную реализацию при классическом рендеринге с помощью растеризации треугольников, например достоверные тени, эффект глубины резкости,	видеокарты не позволяет единовременно принять столько данных за очень короткое время, из-за чего возникает эффект пропажи изображения на несколько кадров. Так как библиотека будет рассчитана на использование с постоянно изменяющимися сценами, потоковая передача данных не будет использована. 3) Не поддерживает встроенные видеокарты. Встроенные в современные процессоры способы ускорения обработки графики имеют большой спрос у пользователей портативных устройств, например ноутбуков. Несмотря на то, что встроенные видеоускорители несомненно слабее дискретных прямых аналогов, они обладают достаточной производительностью для таких повседневных задач как воспроизведение видео в высоком разрешении, либо запуск не сильно требовательных к видеокарте компьютерных игр. В случае реализации опциональной поддержки видеоускорения, библиотека не должна обходить стороной такой доступный широким массам функционал.

Продолжение таблицы 1

Рендерер	Достоинства	Недостатки
	шевеленку, окружающее затенение и непрямое освещение. Трассировка пути это действенный метод для создания фотореалистичных отображений, но качество изображений такого уровня не является целью разрабатываемой библиотеки.3) Поддерживает потоковую передачу данных в видеопамять. SSBO (Shader Storage Buffer Object) это тип данных, добавленный в один из последних стандартов графического интерфейса OpenGL версии 4.3, то есть для видеокарт с датой выпуска после 2012 года. SSBO предназначены для записи в видеокарту произвольных данных большого размера, стандарт гарантирует максимальный объем до 128 мегабайт видеопамяти, однако большинство реализаций от производителей видеокарт позволяет занимать всю доступную видеопамять. Этот функционал используется для хранения воксельного пространства напрямую в видеопамяти, что позволяет экономить на	

Окончание таблицы 1

Рендерер	Достоинства	Недостатки
	транспортировке данных между оперативной и видеопамятью. Для еще большей экономии, в видеопамять отправляется не вся сцена, а только потенциально видимые ее части. В будущей библиотеке использование новейшего функционала библиотеки OpenGL не планируется, так как это привязанность к конкретной реализации взаимодействия с видеокартой, и к тому же сужает аудиторию пользователей до людей с актуальной конфигурацией персонального компьютера.	

1.4. Актуальность проблемы

Выпускная квалификационная работа будет представлять проект по разработке графической библиотеки для создания изображений на основе воксельных алгоритмов. Цель проекта – получение двухмерного изображения воксельной модели при помощи разработки графической библиотеки.

Особенность библиотеки будет заключаться в отсутствии требований к данным, специфичных для какой-либо области. Конвертация данных из используемого в предметной области формата, в простой, понятный библиотеке формат воксельных данных будет осуществляться программистом – пользователем библиотеки.

Архитектура библиотеки будет ориентирована на максимальную простоту в использовании специалистом, что будет выделять ее на фоне аналогов, стремящихся просто предоставить рабочий функционал. Для достижения этой цели библиотека будет иметь следующие особенности:

1) Один заголовочный файл на языке программирования Си.

2) Отсутствие зависимостей от сторонних библиотек.

3) Отсутствие необходимости в комплексных системах сборки, таких как Make, CMake, MSBuild, Ninja, и другие.

4) Набор функций, необходимых для получения изображения, минимизирован.

Для создания изображений библиотека будет использовать алгоритм трассировки лучей. Это позволит реализовать поддержку графических эффектов, которые сложно реализуемы в полигональных рендерерах:

1) Отражения.

2) Точечных источников света.

3) Повоксельных теней.

4) Прозрачности и полупрозрачности.

Библиотека будет способна выполняться на центральном процессоре в один поток или в несколько потоков с помощью стандарта OpenMP, а также, при наличии соответствующей поддержки, на плате видеоускорения с помощью OpenMP.

Библиотека не будет зависеть от реализации взаимодействия с экраном устройства, на котором она работает, то есть рендеринг будет производиться на абстрактный экран произвольного разрешения, что позволит запускаться библиотеке на устройствах без поддержки графических приложений, либо вовсе без какого-либо экрана, что обеспечит возможность трансформировать изображения в аналоговые сигналы, либо сразу организовать распечатку полученного графического изображения.

Библиотека не будет иметь ограничений на размер воксельной сетки, а виртуальная камера будет поддерживать 6 степеней свободы, то есть она сможет совершать движение вперед/назад, влево/вправо, вверх/вниз, а также совершать повороты Эйлера вокруг каждой из трёх взаимно перпендикулярных осей («рыскание», «тангаж», «крен»).

В качестве опционального режима будет реализована техника шахматного рендеринга, сокращающая количество пикселей, закрашенных за один кадр, что разительно ускоряет скорость работы программы, при минимальном воздействии на качество генерируемых изображений.

2. РАЗРАБОТКА АЛГОРИТМОВ И СТРУКТУР ДАННЫХ, ПОЗВОЛЯЮЩИХ ОСУЩЕСТВИТЬ РЕНДЕРИНГ ВОКСЕЛЬНЫХ МОДЕЛЕЙ

2.1. Функциональные требования

Библиотека должна содержать программную реализацию следующего функционала:

1) Создание двухмерного изображения на основе воксельной модели методом трассировки лучей.

2) Учет фонового освещения.

3) Учет диффузного освещения.

4) Учет излучающих поверхностей.

2.2. Нефункциональные требования

К нефункциональным требованиям можно отнести:

1) Один заголовочный файл на языке программирования Си.

2) Функцию установки цвета пикселя на экран должен определять пользователь библиотеки.

3) Возможность самостоятельно определить формат вещественных чисел.

4) Отсутствие динамического выделения памяти в куче в процессе рендеринга.

5) Наличие настроек для подбора оптимального соотношения производительности и качества изображения.

6) Отсутствие внешних зависимостей.

7) Зависимость от функционала стандартной библиотеки языка Си должна быть минимизирована.

2.3. Структура библиотеки

Библиотека представляет собой набор структур и функций, необходимых для реализации алгоритма трассировки лучей через воксельную сеть.

Для удобства восприятия пользователем, исходный код библиотеки будет условно разделен на несколько логических блоков:

1) Блок конфигурации.

В этом блоке будут находиться директивы препроцессора, отвечающие за общие параметры библиотеки при компиляции, например формат вещественных чисел, использование многопоточного режима и другое.

2) Системный блок.

Этот блок будет содержать простейшую реализацию буфера кадров, чтобы разработчик под распространенные на настоящий момент конфигурации компьютеров мог немедленно приступить к работе, без траты времени на создание своей реализации такого функционала.

3) Блок конвертации данных.

Этот блок будет содержать вспомогательные функции, предназначенные для конвертации данных из одного типа в другие, например упаковка цвета из формата 3 байтов на каждый компонент цвета в кодировке RGB, в 32-битное беззнаковое число.

4) Блок математических операций.

В этом блоке будут содержаться функции и макросы для работы с вещественными числами с фиксированной запятой, определения таких структур данных как двухмерные и трехмерные векторы, а также функции для манипуляций над этими векторами.

5) Блок рендеринга.

В этом блоке будут находиться определения структур для непосредственного рендеринга, такие как камера, материал, воксельная сеть и пиксель. Реализация алгоритма трассировки лучей также находится здесь.

2.4. Достижение кроссплатформенности.

В настоящее время существует большое разнообразие аппаратным платформ и операционных систем, поэтому для максимального охвата пользователей библиотека должна поддерживать как можно большее их число. Кроссплатформенность будет реализована путем упрощения портируемости, то есть уменьшением числа жестких требований к аппаратной и программным частям.

Кроссплатформенность разрабатываемой библиотеки будет достигнута следующими способами:

1) Использование языка Си.

Язык программирования Си является одним из самых распространенных языков в мире. Далеко не все платформы имеют реализацию таких высокоабстрактных языков как Java, C# или Python, а в области микроконтроллеров и встраиваемых систем, Си порой является единственной доступной опцией.

2) Весь исходный код находится в одном заголовочном файле.

Такой формат позволяет избежать серьезных проблем с упаковкой, распространением и использованием библиотеки. Для использования функционала нет нужды отдельной компиляции библиотеки как статического или динамического модуля, нет необходимости в использовании комплексных систем сборки таких как CMake или MSBuild.

3) Отсутствие внешних зависимостей.

Библиотека не будет подключать сторонние модули, так как требуемой библиотеки может не оказаться на используемой системе, либо сторонний модуль может просто не поддерживаться. Весь необходимый код, например математические операции, находится в самом исходном коде разрабатываемой библиотеки.

4) Абстрагирование функций для работы с вещественными числами.

Вещественные числа с плавающей запятой являются очень сложной для восприятия структурой, стандарты аппаратной реализации таких чисел занимают не менее полусотни страниц, а сам стандарт содержит неочевидный концепции, такие как NaN, положительный и отрицательные нули. Формат вещественных чисел с фиксированной запятой является более интуитивно понятным, ему не нужен дополнительный вычислительный сопроцессор, который есть не на всех устройствах, этот формат поддерживается везде, где есть поддержка обычных натуральных чисел, а также не содержит погрешности и утери точности,

которыми известны числа с плавающей запятой. Тем не менее, пользователю дана возможность самостоятельно определить формат чисел, которыми будет оперировать библиотека.

5) Процесс непосредственного вывода изображения возложен на пользователя.

Библиотека будет требовать предварительного определения особой функции, отвечающей за размещение пикселя. Это позволит производить рендеринг на экраны любой конструкции, и даже в текстовые командные интерфейсы и файлы. По сути, процесс рендеринга ничем не ограничен, библиотеке не требуется поддержка современных графических интерфейсов, и даже наличие экрана не является необходимостью.

Таким образом, для использования библиотеки необходим только компилятор языка Си.

3. РЕАЛИЗАЦИЯ РАЗРАБОТАННЫХ АЛГОРИТМОВ И СТРУКТУР ДАННЫХ

На основе разработанной структуры из функциональных блоков был реализован следующий функционал библиотеки:

1) Блок конфигурации

Блок конфигурации состоит из директив текстового препроцессора, определяющих поведение остальной части библиотеки, и предварительного объявления функций библиотеки. Описание директив приведено в таблице 2.

Таблица 2 – Директивы текстового препроцессора блока конфигурации

#include <stdlib.h> #include <stdint.h>	Включение элементов стандартной библиотеки языка Си. Файл stdlib.h используется в функциях инициализации буфера кадров. Файл stdint.h необходим для использования типов гарантированного размера.
#if LEVR_USE_OMP #include <omp.h> #endif	Если пользователь библиотеки определил в своей программе макрос LEVR_USE_OMP, то функционал OpenMP будет задействован библиотекой для распараллеливания процесса рендеринга.

Продолжение таблицы 2

#define LEVR_REAL double	Эти директива отвечает за определение типа вещественных чисел, по умолчанию используются вещественные числа с плавающей запятой двойной точности.
#define LEVR_CLAMP(X, lo, hi) (X<lo ? lo : X>hi ? hi : X) #define LEVR_MIN(a, b) ((a) > (b) ? (b) : (a)) #define LEVR_MAX(a, b) ((a) < (b) ? (b) : (a)) #define LEVR_ABS(a) ((a) > 0 ? (a) : -(a))	Эти директивы реализуют общие операции над числами: ограничение в диапазоне, выбор максимального и минимального чисел, нахождение модуля числа.
#define LEVR_mul(a, b) (a * b) #define LEVR_div(a, b) (a / b) #define LEVR_floor(a) floor(a) #define LEVR_sqrt(a) sqrt(a) #define LEVR_pow(a, b) pow(a, b) #define LEVR_PI 3.1415926 #define LEVR_sin(a) sin(a) #define LEVR_cos(a) cos(a) #define LEVR_tan(a) tan(a)	Определение операций умножения и деления для двух чисел, округления вещественного числа, нахождения квадратного корня и возведения в степень. Константное значение числа Пи, определение используемых тригонометрических функций.

2) Системный блок

Согласно проекту, этот блок содержит реализацию буфера кадров, используемую для хранения результатов рендеринга. Описание содержания этого блока дано в таблице 3.

Таблица 3 – Структуры и функции системного блока библиотеки.

struct _LEVR_Screen{
uint32_t* pixels;
uint32_t width;
uint32_t height;
};
	Структура буфера кадров, состоит из указателя на адрес в памяти первого элемента массива беззнаковых целых чисел, представляющих из себя цвета пикселей, а также длины и ширины этого буфера.

void LEVR_screen_flush(LEVR_Screen* screen, uint32_t color){
memset(screen->pixels, color, sizeof(screen->pixels)*screen->width*screen->height);
}
	Функция очистки буфера кадров, заполняет все пиксели указанным цветом.

void LEVR_screen_resize(LEVR_Screen* screen, uint32_t width, uint32_t height){
screen->width = width;
screen->height = height;
screen->pixels = (uint32_t*)malloc(sizeof(uint32_t*) * screen->width*screen->height);
}
	Функция изменения размера буфера кадров, также используется для его первоначальной инициализации.

3) Блок конвертации данных

Этот блок содержит функцию для упаковки цвета из формата отдельных восьми бит на каждый элемент в одно 32-битное число. Реализация представлена в таблице 4.

Таблица 4 - Функции блока конвертации данных.

uint32_t LEVR_rgb_to_u32(uint8_t r, uint8_t g, uint8_t b) {
uint8_t alpha = 255;
return (alpha << 24) + (b << 16) + (g << 8) + r;
}
	Функция конвертации цвета.

4) Блок математических операций

Блок содержит реализацию необходимых для рендеринга функций над трехмерными векторами, а также сами структуры этих векторов. Содержание этого блока представлено в таблице 5.

Таблица 5 - Описание функций блока математических операций.

struct _LEVR_vec3 {
LEVR_REAL x, y, z;
};
struct _LEVR_vec3i {
int32_t x, y, z;
};
	Определение структур трехмерных вещественного и целочисленного векторов.

LEVR_vec3 LEVR_vec3_add(LEVR_vec3 a, LEVR_vec3 b){
LEVR_vec3 r;
r.x = a.x + b.x;
r.y = a.y + b.y;
r.z = a.z + b.z;
return r;
}
	Функция сложения двух векторов, принимает два вектора, возвращает результат их поэлементного сложения.

LEVR_vec3 LEVR_vec3_sub(LEVR_vec3 a, LEVR_vec3 b){
LEVR_vec3 r;
r.x = a.x - b.x;
r.y = a.y - b.y;
r.z = a.z - b.z;
return r;
}
	Функция разности двух векторов, принимает два вектора, возвращает результат их поэлементного вычитания.

LEVR_vec3 LEVR_vec3_scale(LEVR_vec3 a, LEVR_vec3 b){
LEVR_vec3 r;
r.x = LEVR_mul(a.x, b.x);
r.y = LEVR_mul(a.y, b.y);
r.z = LEVR_mul(a.z, b.z);
return r;
}
	Функция масштабирования вектора на вектор, принимает два вектора, возвращает результат их поэлементного умножения.

LEVR_vec3 LEVR_vec3_scalef(LEVR_vec3 a, LEVR_REAL b){ LEVR_vec3 r;	Функция масштабирования вектора на число, принимает

Продолжение таблицы 5

r.x = LEVR_mul(a.x, b);
r.y = LEVR_mul(a.y, b);
r.z = LEVR_mul(a.z, b);
return r;
}
	масштабируемый вектор и вещественное число, возвращает результат поэлементного умножения вектора на это число.

LEVR_vec3 LEVR_vec3_divf(LEVR_vec3 a, LEVR_REAL b){
LEVR_vec3 r;
if(b == 0) return a;
r.x = LEVR_div(a.x, b);
r.y = LEVR_div(a.y, b);
r.z = LEVR_div(a.z, b);
return r;
}
	Функция деления вектора на число, принимает вектор и вещественное число, возвращает результат поэлементного деления вектора на это число.

LEVR_REAL LEVR_vec3_length(LEVR_vec3 a)
{
LEVR_REAL l = LEVR_sqrt(LEVR_pow(a.x, 2) + LEVR_pow(a.y, 2) + LEVR_pow(a.z, 2));
return l;
}
	Функция нахождения длины указанного вектора.

LEVR_vec3 LEVR_vec3_normalize(LEVR_vec3 a)
{
return LEVR_vec3_divf(a, LEVR_vec3_length(a));
}
	Функция нормализации указанного вектора.

LEVR_REAL LEVR_vec3_dot(LEVR_vec3 a, LEVR_vec3 b)
{
LEVR_REAL r = LEVR_mul(a.x, b.x) + LEVR_mul(a.y, b.y) + LEVR_mul(a.z, b.z);
return r;
}
	Функция поиска скалярного произведения двух векторов.

LEVR_vec3 LEVR_vec3_rotate(LEVR_vec3 a, LEVR_vec3 b){ LEVR_vec3 r; LEVR_REAL pitch = LEVR_div(LEVR_mul(b.x, LEVR_PI), 180); LEVR_REAL yaw = LEVR_div(LEVR_mul(b.y, LEVR_PI), 180); LEVR_REAL roll = LEVR_div(LEVR_mul(b.z, LEVR_PI), 180); LEVR_REAL cosa = LEVR_cos(roll); LEVR_REAL sina = LEVR_sin(roll); LEVR_REAL cosb = LEVR_cos(yaw); LEVR_REAL sinb = LEVR_sin(yaw); LEVR_REAL cosc = LEVR_cos(pitch); LEVR_REAL sinc = LEVR_sin(pitch); LEVR_REAL Axx = LEVR_mul(cosa, cosb); LEVR_REAL Axy = LEVR_mul(LEVR_mul(cosa, sinb), sinc) - LEVR_mul(sina, cosc); LEVR_REAL Axz = LEVR_mul(LEVR_mul(cosa, sinb), cosc) + LEVR_mul(sina, sinc);	Функция поворота вектора на указанные углы Эйлера, принимает вектор, который надо повернуть, а также вектор, состоящий из значений требуемых углов.

Окончание таблицы 5

LEVR_REAL Ayx = LEVR_mul(sina, cosb);
LEVR_REAL Ayy = LEVR_mul(LEVR_mul(sina, sinb), sinc) + LEVR_mul(cosa, cosc);
LEVR_REAL Ayz = LEVR_mul(LEVR_mul(sina, sinb), cosc) - LEVR_mul(cosa, sinc);
 	
LEVR_REAL Azx = -sinb;
LEVR_REAL Azy = LEVR_mul(cosb, sinc);
LEVR_REAL Azz = LEVR_mul(cosb, cosc);
 	
r.x = LEVR_mul(Axx, a.x) + LEVR_mul(Axy, a.y) + LEVR_mul(Axz, a.z);
r.y = LEVR_mul(Ayx, a.x) + LEVR_mul(Ayy, a.y) + LEVR_mul(Ayz, a.z);
r.z = LEVR_mul(Azx, a.x) + LEVR_mul(Azy, a.y) + LEVR_mul(Azz, a.z);
 	
return r;
}	

5) Блок рендеринга

Блок рендеринга является основной и самой обширной частью библиотеки, в нем находятся описания структур и функция для проведения рендеринга.

Реализация структур, описывающих используемые для рендеринга сущности, представлена в таблице 6.

Таблица 6 – Реализация структур блока рендеринга.

struct _LEVR_Pixel{
uint32_t x;
uint32_t y;
uint32_t color;
};
	Структура пикселя, состоит из координат и цвета в формате 32-битного целого числа.

struct _LEVR_Ray{
LEVR_vec3 pos;
LEVR_vec3 dir;
};
	Структура луча, состоит из двух векторов – определяющих позицию и направление.

struct _LEVR_Camera{
uint32_t width;
uint32_t height;
LEVR_vec3 pos;
LEVR_vec3 dir;
LEVR_vec3 up;
uint32_t fov;
};
	Структура виртуальной камеры, состоит из угла пол зрения в градусах, длины и ширины окна просмотра, векторов описывающих положение камеры, направление ее взгляда, а также направление «вверх».

struct _LEVR_Material{ uint32_t color; LEVR_REAL ambient; LEVR_REAL emittance; LEVR_REAL diffuse;	Структура материала вокселя, содержит информацию о цвете, коэффициент диффузного освещения, а также силу

Продолжение таблицы 6.

};
struct _LEVR_MaterialSet{
LEVR_Material* materials;
uint8_t max_materials;
};
	излучения света.
	Структура набора материалов воксельной сетки, представляет из себя пару из массива материалов, и их числа.

struct _LEVR_Grid{
uint8_t* voxels;
uint32_t width;
uint32_t height;
uint32_t depth;
};
	Структура воксельной сети, по умолчанию представляет из себя массив из 256 индексов в массиве материалов, а также размеры сетки.

struct _LEVR_Scene{
LEVR_Grid grid;
LEVR_vec3i* lights;
uint32_t max_lights;
};
	Структура графической сцены, состоит из воксельной сетки, а также массива координат источников света.

struct _LEVR_Hit{
uint8_t material;
uint32_t dist;
LEVR_vec3i pos;
LEVR_vec3 normal;
};
	Структура результата пересечения луча с воксельной сеткой, содержит информацию о материале пересеченного вокселя, количество итераций алгоритма движения луча до пересечения, положение этого вокселя, а также нормаль к его поверхности.

struct _LEVR_State{
uint32_t flags;
void (*set_pixel)(LEVR_State*, LEVR_Pixel*);
};
	Структура состояния рендера, содержит 32-битное число, служащее хранилищем различных состояний рендера, а также указатель на функцию, которая будет вызвана рендерером для установки пикселя на экран.

Непосредственный процесс рендеринга состоит из трех подпрограмм, последовательно выполняющих отдельные этапы построения изображения:

1) Генерация лучей и финальная установка цвета каждого пикселя.

2) Расчет цвета, освещения и теней для каждого пикселя.

3) Поиск пересечения луча с сеткой.

Далее будет рассмотрено проектирование и реализация каждой из этих функций по отдельности.

Функция LEVR_render

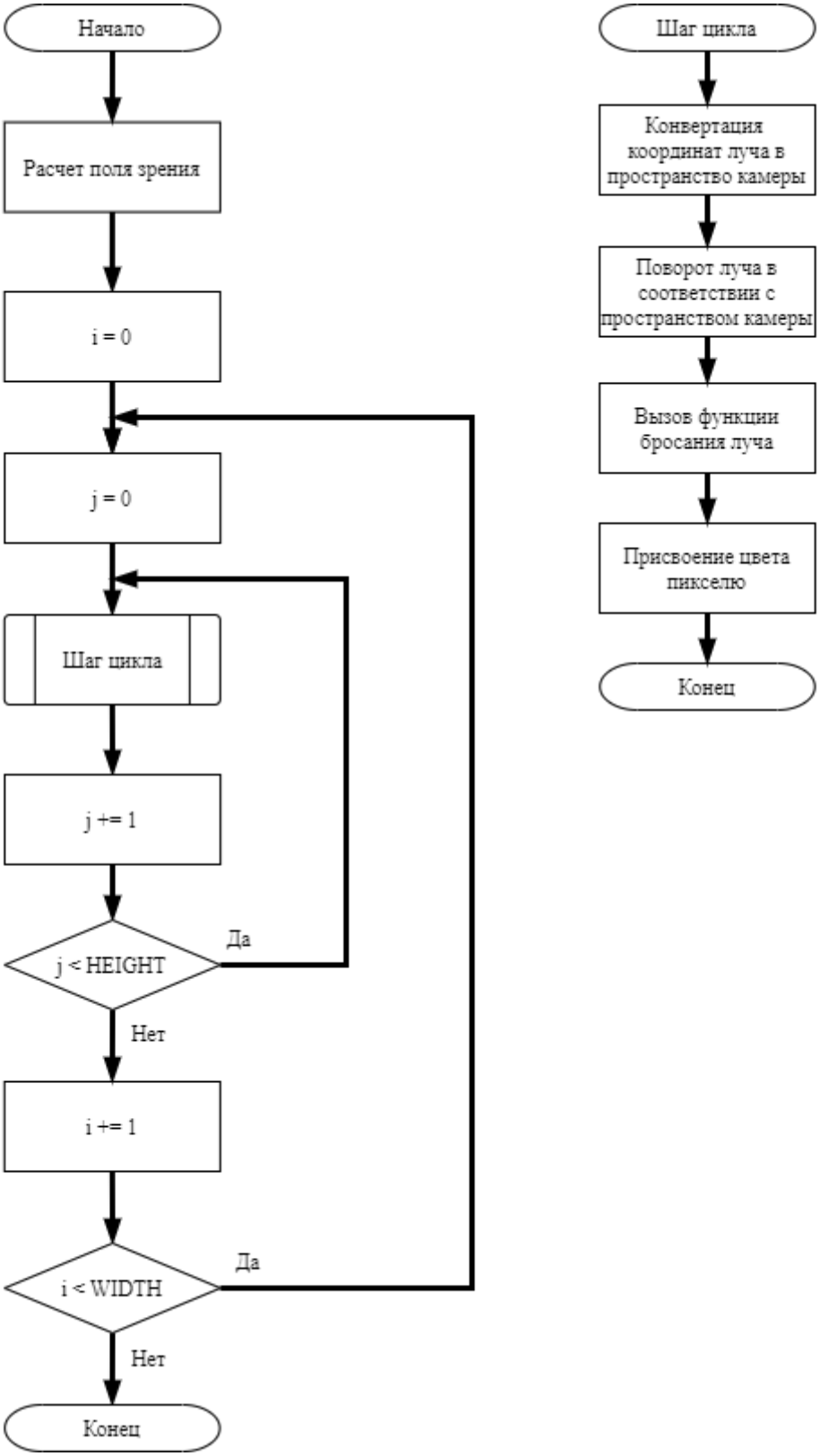

Рисунок 1 – Блок-схема функции LEVR_render

На первом шаге функция производит расчет поля зрения, необходимый для последующей генерации лучей камеры, реализация представлена в листинге 1.

Листинг 1 – Реализация расчета поля зрения

```
   LEVR_REAL arg = LEVR_mul(LEVR_div(camera.fov, 2),
LEVR_div(LEVR_PI, 180));
   LEVR_REAL tang = LEVR_tan(arg);
   LEVR_REAL FOV = LEVR_div(camera.width, LEVR_mul(tang, 2));
   LEVR_REAL aspectratio = LEVR_div(camera.width, camera.height);
```

После чего начинается основной цикл рендеринга, в котором для каждого пикселя рассчитывается соответствующий луч и вызывается функция бросания этого луча, реализация описана в листинге 2.

Листинг 2 – Реализация основного цикла рендеринга

```
   #ifdef LEVR_USE_OMP
     #pragma omp parallel for private(px, py, pixel, ray, dir)
   #endif
   for(int i = 0; i < camera.width; i++){
          #if LEVR_CHECKERBOARD_RENDERING
               flag = !flag;
          #endif
          for(int j = 0; j < camera.height; j++){
                    #if LEVR_CHECKERBOARD_RENDERING
                         flag = !flag;
                         if (flag) continue;
                    #endif
                    pixel.x = i;
                    pixel.y = j;
                    px = LEVR_mul(LEVR_mul((LEVR_mul(2,
(LEVR_div((i + 0.5), camera.width))) - 1), FOV), aspectratio);
                    py = LEVR_mul((1 - LEVR_mul(2, (LEVR_div((j +
0.5), camera.height)))), FOV);
                    dir = LEVR_vec3_rotate((LEVR_vec3){px, py,
camera.fov}, camera.dir);
                    ray = (LEVR_Ray){camera.pos, dir};
                    pixel.color = LEVR_cast(ray, scene, ms);
                    state->set_pixel(&state, &pixel);
          }
   }
   #if LEVR_CHECKERBOARD_RENDERING
     state->flags ^= (1 << 0);
   #endif
```

Макросы LEVR_USE_OMP и LEVR_CHECKERBOARD_RENDERING определяются пользователем вручную в своей программе, если они определены – то они модифицируют основной цикл для увеличения производительности. LEVR_USE_OMP включает директиву стандарта OpenMP для выполнения цикла в параллельном режиме. LEVR_CHECKERBOARD_RENDERING обеспечивает функционал, который прерывает половину итераций цикла, из-за чего изображение становится похожим на шахматную доску. В следующем кадре

прерывается другая половина итераций цикла, формируя полное изображение за 2 кадра. Эта техника сокращает количество необходимых расчетов, при этом ее применение практически незаметно человеческому глазу при достаточном количестве кадров в секунду.

Функция LEVR_cast

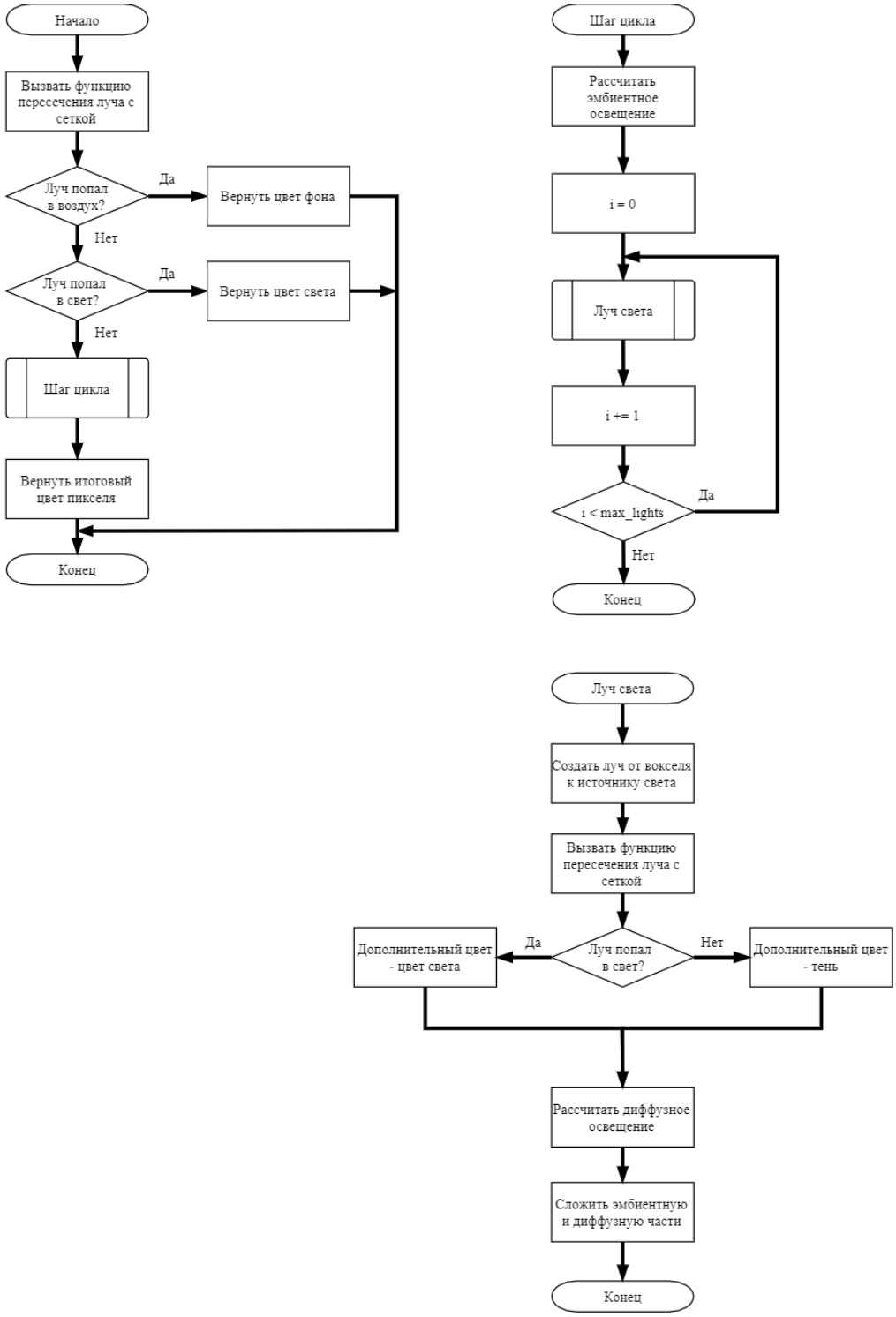

Рисунок 2 – Блок-схема функции LEVR_cast

Реализация этой блок-схемы представлена в листинге 3.

Листинг 3 – Реализация функции LEVR_cast

```
   uint32_t LEVR_cast(LEVR_Ray ray, LEVR_Scene scene,
LEVR_MaterialSet ms){
     LEVR_Hit hit = LEVR_intersect(ray, scene.grid, 512);
     LEVR_vec3 light_pos = {0};
     LEVR_vec3 hit_pos = {0};
     LEVR_vec3 res = {0};
     int i;
     if (hit.material == 0) return 0xBBBF00;
     if (ms.materials[hit.material].emittance > 0) return
ms.materials[hit.material].color;
     LEVR_REAL ambient = hit.material.ambient;
     uint8_t red = (ms.materials[hit.material].color >> 16) &
0xff;
     uint8_t green = (ms.materials[hit.material].color >> 8) &
0xff;
     uint8_t blue = ms.materials[hit.material].color & 0xff;
     LEVR_vec3 color = {0};
     color.x = blue;
     color.y = green;
     color.z = red;

     res = LEVR_vec3_add(res, LEVR_vec3_scalef(color, ambient));
     for (i = 0; i < scene.max_lights; i++){

     light_pos.x = scene.lights[i].x + 0.5;
     light_pos.y = scene.lights[i].y + 0.5;
     light_pos.z = scene.lights[i].z + 0.5;
     hit_pos.x = hit.pos.x + hit.normal.x;
     hit_pos.y = hit.pos.y + hit.normal.y;
     hit_pos.z = hit.pos.z + hit.normal.z;

     LEVR_vec3 light_dir =
LEVR_vec3_normalize(LEVR_vec3_sub(light_pos, hit_pos));
     LEVR_Ray light_ray = {hit_pos, light_dir};
     LEVR_Hit light_hit = LEVR_intersect(light_ray, scene.grid,
512);
     LEVR_REAL emit = ms.materials[light_hit.material].emittance;
     LEVR_vec3 light_color;
     if(emit > 0){
          uint8_t lred = (ms.materials[light_hit.material].color
>> 16) & 0xff;
          uint8_t lgreen =
(ms.materials[light_hit.material].color >> 8) & 0xff;
          uint8_t lblue = ms.materials[light_hit.material].color
& 0xff;
          light_color = (LEVR_vec3){lred / 255, lgreen / 255,
lblue / 255};
     } else {
```

Продолжение листинга 3

```
            light_color = (LEVR_vec3){emit, emit, emit};
        }

    LEVR_REAL diffuse =
LEVR_MAX(LEVR_vec3_dot(LEVR_vec3_normalize(hit.normal),
light_dir), hit.material.diffuse);
        res = LEVR_vec3_add(res,
LEVR_vec3_scalef(LEVR_vec3_scale(color, light_color), diffuse));
        }
        return LEVR_rgb_to_u32(LEVR_CLAMP(res.x, 0, 0xFFFFFF),
LEVR_CLAMP(res.y, 0, 0xFFFFFF), LEVR_CLAMP(res.z, 0, 0xFFFFFF));
    }
```

Данная подпрограмма производит расчет цвета пикселя. Для этого в начале выполнения вызывается функция поиска пересечения луча с воксельной сеткой, материал, являющийся результатом выполнения функции, дает необходимые для дальнейших расчетов значения – цвет и коэффициенты для моделирования освещенности. Дополнительно производятся проверка результата на коэффициент излучаемости, так как в данной модели не происходит расчета теней от источника освещения, а также проверка на выход луча за границу сетки или превышение дальности видимости, что приводит к окрашиванию пикселя в цвет фона, также без произведения расчета освещенности. После успешного нахождения непустого неизлучающего пересечения начинается второй этап определения цвета, в котором происходит выпускание дополнительных лучей из точки пересечения первоначального луча во все известные источники света на сцене. Если вторичный луч на своем пути встретил воксель, не излучающий свет, то этот воксель становится затененным, в противном случае происходит осветление пикселя в соответствии с коэффициентом диффузного освещения встреченного первоначальным лучом материала. Полученные результаты от всех источников света складываются, что и представляет собой итоговый цвет пикселя.

Функция LEVR_intersect

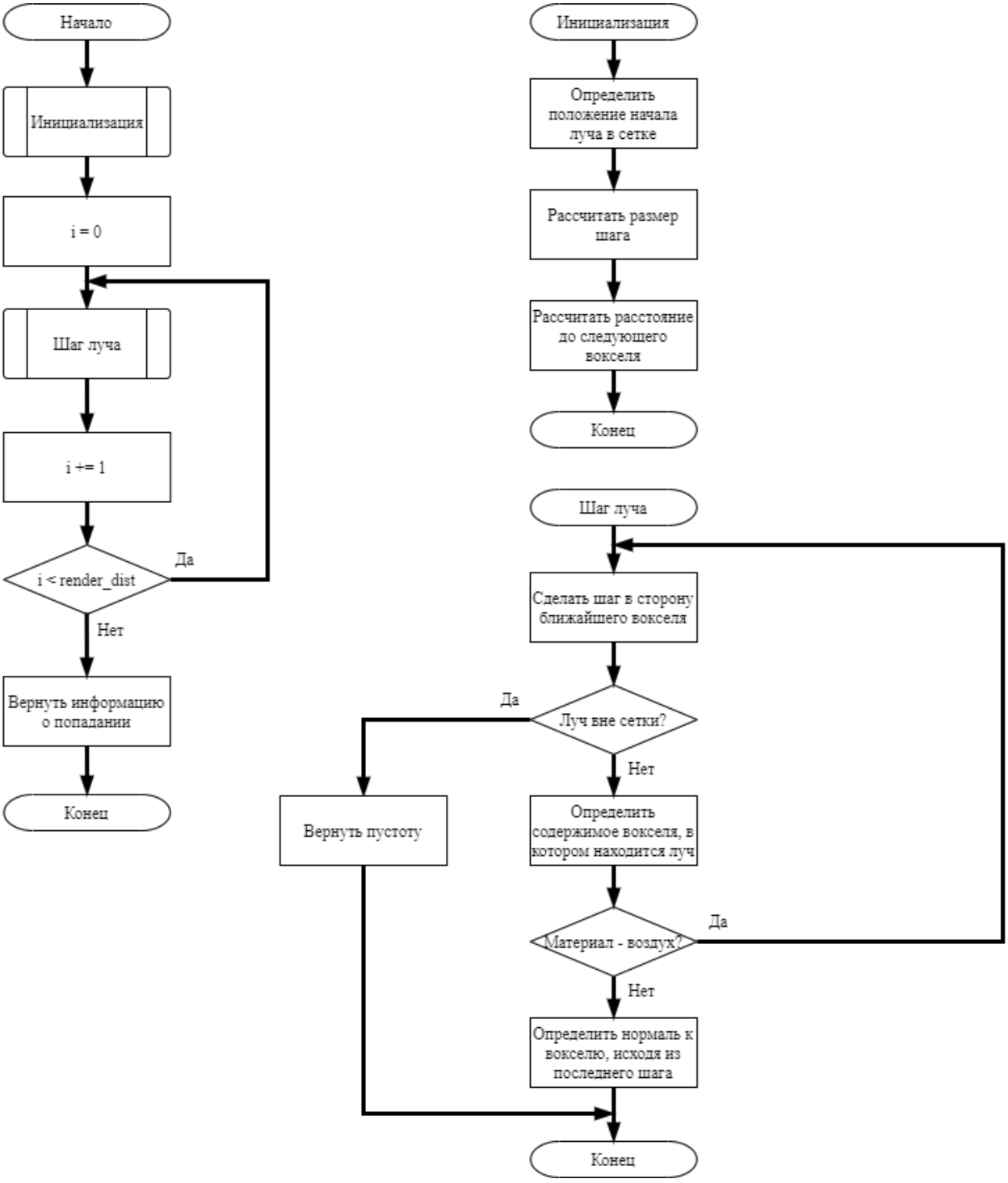

Рисунок 3 – Блок-схема функции LEVR_intersect

Программный код соответствующий этой схеме представлен в листинге 4.

Листинг 4 – Реализация функции LEVR_intersect

```
   LEVR_Hit LEVR_intersect(LEVR_Ray ray, LEVR_Grid grid, uint32_t
renderdist){
     int i = 0;
     int index = 0;
     uint8_t voxel = 0;
     int side = 0;
     LEVR_vec3 coord = {LEVR_floor(ray.pos.x),
LEVR_floor(ray.pos.y), LEVR_floor(ray.pos.z)};
     int x, y, z;
     LEVR_vec3 delta = {1 / LEVR_ABS(ray.dir.x), 1 /
LEVR_ABS(ray.dir.y), 1 / LEVR_ABS(ray.dir.z)};
     LEVR_vec3 step = {LEVR_mul(ray.dir.x, delta.x),
LEVR_mul(ray.dir.y, delta.y), LEVR_mul(ray.dir.z, delta.z)};
     LEVR_vec3 tmax = {0, 0, 0};
     LEVR_vec3 normal = {0, 0, 0};
     LEVR_Hit hit;
     hit.pos = (LEVR_vec3i){0, 0, 0};
     hit.normal = normal;
     hit.material = 0;
     hit.dist = 0;

     if(ray.dir.x < 0){
          tmax.x = LEVR_mul((ray.pos.x - (coord.x)), delta.x);
     } else {
          tmax.x = LEVR_mul((1.0F - (ray.pos.x - (coord.x))),
delta.x);
     }
     if(ray.dir.y < 0){
          tmax.y = LEVR_mul((ray.pos.y - (coord.y)), delta.y);
     } else {
          tmax.y = LEVR_mul((1.0F - (ray.pos.y - (coord.y))),
delta.y);
     }
     if(ray.dir.z < 0){
          tmax.z = LEVR_mul((ray.pos.z - (coord.z)), delta.z);
     } else {
          tmax.z = LEVR_mul((1.0F - (ray.pos.z - (coord.z))),
delta.z);
     }


     while ( i < renderdist && voxel == 0 ){
          if (tmax.x < tmax.y && tmax.x < tmax.z) {
               coord.x += step.x;
               tmax.x += delta.x;
               side = 0;
          } else if (tmax.y < tmax.z) {
               coord.y += step.y;
               tmax.y += delta.y;
               side = 1;
          } else {
```

Продолжение листинга 4

```
                coord.z += step.z;
                tmax.z += delta.z;
                side = 2;
            }
            x = ceil(coord.x);
            y = ceil(coord.y);
            z = ceil(coord.z);

            if(x >= grid.width || x < 0){
                return hit;
            }
            if(y >= grid.height || y < 0){
                return hit;
            }
            if(z >= grid.depth || z < 0){
                return hit;
            }

            index = x + y * grid.height + z * grid.width *
grid.height;
            if(index >= grid.width * grid.height * grid.depth ||
index < 0){
                return hit;
            } else {
                voxel = grid.voxels[index];
            }
            i++;
        }
        if(i < renderdist && voxel != 0){
            if(side == 0){
                if(ray.dir.x < 0){
                    normal.x = 1;
                } else {
                    normal.x = -1;
                }
                normal.y = 0;
                normal.z = 0;
            }
            else if (side == 1){
                if(ray.dir.y < 0){
                    normal.y = 1;
                } else {
                    normal.y = -1;
                }
                normal.x = 0;
                normal.z = 0;
            }
            else{
                if(ray.dir.z < 0){
                    normal.z = 1;
                } else {
```

Окончание листинга 4

```
            normal.z = -1;
            }
            normal.x = 0;
            normal.y = 0;
        }
        hit.pos.x = x;
        hit.pos.y = y;
        hit.pos.z = z;
        hit.material = voxel;
        hit.normal = normal;
        hit.dist = i;
        return hit;
  } else {
        return hit;
  }
}
```

Алгоритм движения луча состоит из 3 частей: инициализация переменных, движение луча и формирование результата.

Длина шага по каждой компоненте координат инициализируется таким образом, чтобы за одну итерацию перейти сразу к следующей ячейке сетки. Такой подход позволяет уменьшить количество необходимых итераций, а также устраняет проблему «проскальзывания» луча, при которой луч не задевает непустой воксель из-за того, что он находится между двумя точками остановки луча.

Так как воксели в этой реализации не представляют из себя комплексные трехмерные поверхности, определить нормаль к встреченному лучом вокселю можно избежав комплексных расчетов, путем простого определения стороны, в которую попал луч, чего вполне достаточно для реализации повоксельного освещения.

4. ПРОВЕДЕНИЕ ТЕСТИРОВАНИЯ РЕАЛИЗОВАННЫХ АЛГОРИТМОВ

Для проведения тестирования была создана программа с использованием библиотеки app.h, содержащей простейшую кроссплатформенную реализацию окна и обработчика ввода.

Тестовая программа состоит из нескольких функций, последовательно подготавливающих воксельную сцену для рендеринга.

Функция CreateGrid читает файл с воксельной моделью и загружает ее в память, ее реализация представлена в листинге 5.

Листинг 5 – Реализация функции CreateGrid

```
void CreateGrid()
{
  grid.width = 256;
  grid.height = 256;
```

Продолжение листинга 5

```
  grid.depth = 256;
  grid.voxels = malloc(grid.width * grid.height * grid.depth *
sizeof(uint8_t));
  uint8_t *buf = malloc(255 * 255 * 255 * sizeof(uint8_t));

  FILE *file = fopen("sponza.raw", "r");
  int c;
  int i = 0;
  int j = 0;
  int k = 0;
  while ((c = fgetc(file)) != EOF)
  {
       buf[i] = c - '0';
       i++;
  }
  fclose(file);
  for(i = 0; i < 255; i++){
       for(j = 0; j < 255; j++){
            for(k = 0; k < 255; k++){
                 grid.voxels[(k)*grid.width*grid.height +
(j)*grid.height + (i)] = buf[k*255*255 + j*255+i];
            }
       }
  }
  free(buf);

  grid.voxels[(69)*grid.width*grid.height + (10)*grid.height +
(50)] = 4;
}
```

Функция InitMaterials создает набор материалов, состоящий из пустого пространства, красных, синих и зеленых вокселей, и белого источника света, ее реализация представлена в листинге 6.

Листинг 6 – Реализация функции InitMaterials

```
void InitMaterials(){
  ms.max_materials = 5;

  LEVR_Material air = {0};
  air.color = LEVR_rgb_to_u32(0, 0, 0);

  LEVR_Material red = {0};
  red.color = LEVR_rgb_to_u32(100, 0, 0);
  red.emittance = 0;
  red.ambient = 0.5;
  red.diffuse = 0;
```

Продолжение листинга 6

```
    LEVR_Material green = {0};
    green.color = LEVR_rgb_to_u32(0, 100, 0);
    green.emittance = 0;
    green.ambient = 0.5;
    green.diffuse = 0;

    LEVR_Material blue = {0};
    blue.color = LEVR_rgb_to_u32(0, 0, 100);
    blue.emittance = 0;
    blue.ambient = 0.5;
    blue.diffuse = 0;

    LEVR_Material light = {0};
    light.color = 0xFFFFFF;
    light.emittance = 1.0;

    ms.materials = malloc(sizeof(LEVR_Material) *
ms.max_materials);
    ms.materials[0] = air;
    ms.materials[1] = red;
    ms.materials[2] = green;
    ms.materials[3] = blue;
    ms.materials[4] = light;

}
```

Функция CreateCamera инициализирует структуру камеры и устанавливает ее в первоначальное положение на сцене, реализация представлена в листинге 7.

Листинг 7 – Реализация функции CreateCamera

```
void CreateCamera(){
  cam.width = SCWIDTH;
  cam.height = SCHEIGHT;

  LEVR_vec3 pos = {9, 9, 9};
  LEVR_vec3 dir = {0.1, 0.1, 0};
  LEVR_vec3 up = {0, 1, 0};
  cam.pos = pos;
  cam.dir = dir;
  cam.up = up;
  cam.fov = 70;
}
```

После чего программа в цикле производит рендеринг изображения и вывод его на экран, останавливаясь по запросу пользователя. Для упрощения тестирования, в программе реализовано считывание нажатий клавиш клавиатуры и перемещений мыши, для управления виртуальной камерой.

Полученные с помощью библиотеки изображения на основе воксельных моделей представлены на рисунках:

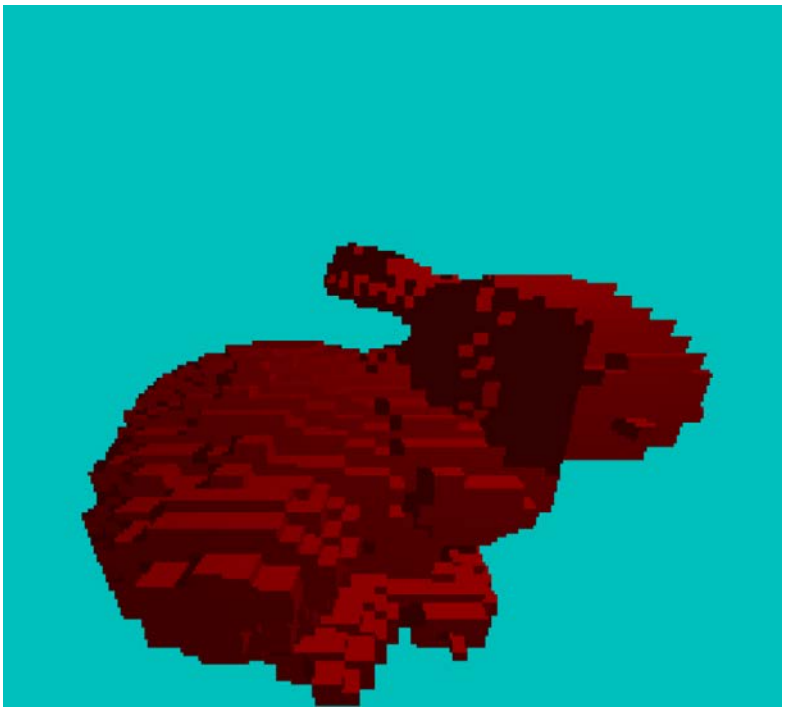

Рисунок 4 – Модель «Стенфордский кролик», 48x48x48 вокселей.

Рисунок 5 – Оригинальная модель «Стенфордский кролик», 69451 полигон.

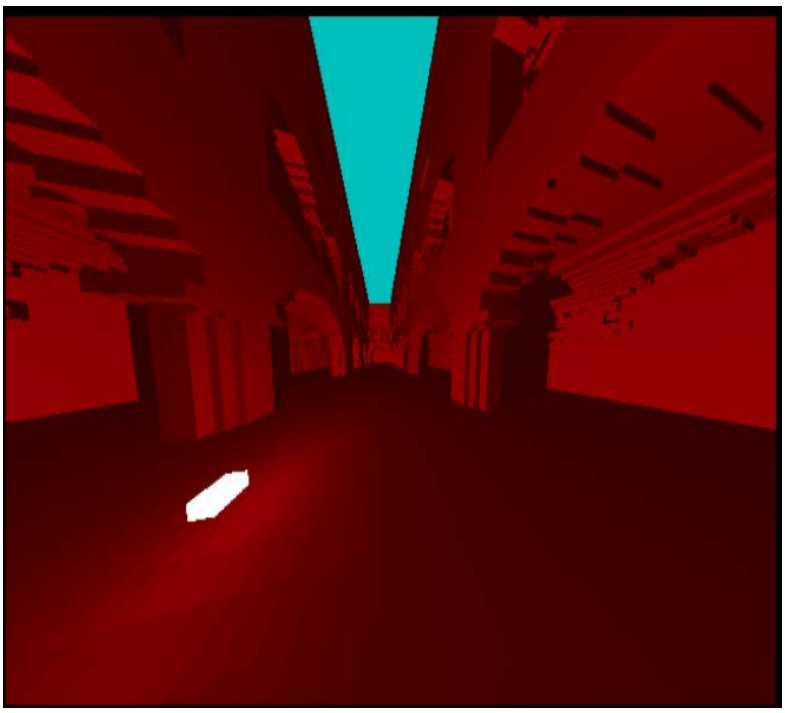

Рисунок 6 – Модель «Атриум дворца Спонца», 512x512x512 вокселей.

Рисунок 7 – Та же модель под другим ракурсом.

Рисунок 8 – Оригинальная модель «Атриум дворца Спонца», 66450 полигонов.

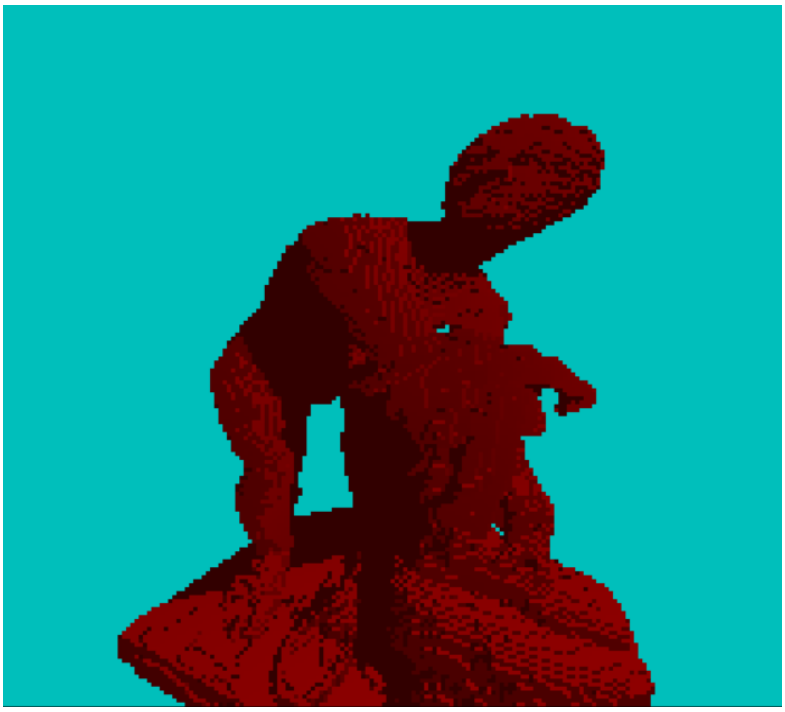

Рисунок 9 – Модель «Гермес, завязывающий сандалию», 256x256x256 вокселей.

Рисунок 10 – Оригинальная модель «Гермес, завязывающий сандалию», 2000022 полигона.

В результате тестирования были проверены основные функции библиотеки, итоги которого соответствуют цели проекта - получение двухмерного изображения воксельной модели при помощи разработки графической библиотеки. Для достижения представленной цели были выполнены следующие задачи:

1. Аналитический обзор научно-технической, нормативной и методической литературы по тематике работы;
2. Разработка алгоритмов и структур данных, позволяющих осуществить рендеринг воксельных моделей;
3. Реализация разработанных алгоритмов и структур данных;
4. Проведение тестирования реализованных алгоритмов.

ЗАКЛЮЧЕНИЕ

Компьютерная графика позволяет использовать современные технология для визуализации объектов различной типологии, что делает возможным применять ее алгоритмы в различных направлениях науки и техники. К одним из востребованных функциональных возможностей таких технологий можно отнести воксельный метод для рендеринга объектов, реализация которого выполнена в виде кроссплатформенной графической библиотеки.

В первой главе выпускной квалификационной работы был проведен аналитический обзор научно-технической, нормативной и методической литературы по тематике работы, перечислены существующие аналоги, на основе которых определена актуальность работы.

Во второй главе были разработаны алгоритмы и структуры данных, позволяюще осуществить рендеринг воксельных моделей, определены функциональные и нефункциональные требования, созданы архитектурные решения, обеспечивающие кроссплатформенность библиотеки.

В третьей главе описана реализация разработанного ранее функционала, с приведены блок-схемы основных алгоритмов библиотеки, с комментариями по реализации отдельных элементов программы.

В четвертой главе разработано тестовое приложение, с помощью которого были получены корректные двухмерные изображения на основе трехмерных воксельных моделей.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Mileff, P. The Past and the Future of Computer Visualization. / P. Mileff, J. Dudra // Production Systems and Information Engineering. – 2022. – V. 10, №1. – P. 16–29.

2. Tyc, J. A Scoping review of voxel-model applications to enable multi-domain data integration in architectural design and urban planning. / T. Selami, D.S. Hensel, M. Hensel. // Architecture. – 2023. – V. 3, №3 (2) – P. 137–174.

3. Srihari, S. N. Representation of Three-Dimensional Digital Images. // ACM Comput. Surv. – 1981. – V. 13, №4 – P. 399–424.

4. Lorensen, W. Marching cubes: a high resolution 3d surface construction algorithm. / W. Lorensen, H. Cline // ACM SIGGRAPH Computer Graphics. – 1987. – V. 21, №4 – P. 163-169.

5. Raman, S. Quality isosurface mesh generation using an extended marching cubes lookup table. / S. Raman, R. Wenger // Comput. Graph. Forum. – 2008. – V. 27, №3 – P. 791-798.

6. Sarah, F. Constrained elastic surface nets: generating smooth surfaces from binary segmented data. / F. Sarah, F. Gibson // Medical Image Computing and computer-assisted intervention (MICCAI '98) – Berlin: Springer Berlin Heidelberg Publ., 1998. – P. 888–898.

7. Tao, J. Dual contouring of hermite data. / J. Tao, L. Frank, S. Scott, J. Warren // ACM Transactions on Graphics. – 2002. – V. 21, №3 – P. 339–346.

8. Poston, T. Multiresolution isosurface extraction with adaptive skeleton climbing. / T. Poston, T. Wong, P.A. Heng // Computer Graphics Forum. – 1998. – V. 17, №3 – P.137–147.

9. Amanatides, J. A Fast Voxel Traversal Algorithm for Ray Tracing. / J. Amanatides, A. Woo // Proceedings of EuroGraphics. – 1987. – V. 87, №3. – P. 3–10.

10. Eisemann, E. Fast Scene Voxelization and Applications. / E. Eisemann X. Décoret // Proceedings of the 2006 symposium on Interactive 3D graphics and games – New York: Association for Computing Machinery Publ., 2006. – P. 71–78.

11. Gorte, B. A COMPUTATIONALLY CHEAP TRICK TO DETERMINE SHADOW IN A VOXEL MODEL. / B. Gorte, K. Zhou, C. Sande, C. Valk // ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences. – 2018. – V. 4, №4. – P. 67–71.

12. Nichols, G. Interactive, multiresolution image-space rendering for dynamic area lighting. / G. Nichols, R. Penmatsa, C. Wyman // IComputerGraphics Forum. – 2010. – V. 29, №4. – P. 1279–1288.

13. Caon, M. Voxel-based computational models of real human anatomy: a review. // Radiat Environ Biophys. – 2004. – V. 42, №4. – P. 229–235.

14. Nießner, M. Real-time 3D reconstruction at scale using voxel hashing. / M. Nießner, M. Zollhöfer, S. Izadi, M. Stamminger // ACM Trans. Graph. – 2013. – V. 32, №6. – P. 1–11.

15. Beckhaus, S. Hardware-based voxelization for 3d spatial analysis / S. Beckhaus, J. Wind, T. Strothotte // Proceedings of the 5th international conference on computer graphics and imaging. – Canmore, Alberta, Canada, 2002. – V. 20. – P. 15–20.

16. Jørgensen, F. A method for cognitive 3D geological voxel modelling of AEM data //Bulletin of Engineering Geology and the Environment. – 2013. – V. 3, №72. – P. 421–432.

17. Gorte, B. NAVIGATION IN INDOOR VOXEL MODELS / B. Gorte, S. Zlatanova, F. Fadli // ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences. – 2019. – V. 4, №2. – P. 279–283.

18. PND3D Demo and Source Code. – http://advsys.net/ken/voxlap/pnd3d.htm (дата обращения: 28.10.2023).

19. DOOMMY VOXEL ENGINE. – https://github.com/jabberx/doommy (дата обращения: 28.10.2023).

20. DoonEngine a voxel path-tracer. – https://github.com/frozein/DoonEngine (дата обращения: 28.10.2023).