МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**КУРСОВАЯ РАБОТА**

**Моделирование трехмерных сцен на основе оптимизированных октодеревьев**

Работу выполнил                                          Р. В. Реутов

(подпись, дата)                      (инициалы, фамилия)

Факультет компьютерных технологий и прикладной математики 3 курс

Направление 02.03.02 – «Фундаментальная информатика и информационные технологии»

Научный руководитель

доц., к.т.н.                                                                          Т.А. Приходько

    (подпись, дата)                      (инициалы, фамилия)

Нормоконтролер

доц., к.т.н.                                               Е. Е. Полупанова

    (подпись, дата)                  (инициалы, фамилия)

Краснодар

2021

СОДЕРЖАНИЕ

[1 Теоретические положения об использовании деревьев в приложениях компьютерной графики 4](#_Toc73114310)

[1.1 Проблемы, возникающие при разработке трехмерного видеодвижка 4](#_Toc73114311)

[1.2 K-мерные деревья поиска 5](#_Toc73114312)

[1.3 Октодеревья 6](#_Toc73114313)

[1.4. Наивная реализация октодерева 9](#_Toc73114314)

[2 Методы оптимизации алгоритмов на октодеревьях 10](#_Toc73114315)

[2.1 Общая оптимизация октодерева 10](#_Toc73114316)

[2.2 Сравнение производительности и расхода памяти в наивной и оптимизированной реализации 12](#_Toc73114317)

[2.3 Упаковка октодерева в одномерный массив 14](#_Toc73114318)

[3. Реализация октодеревьев на практике 17](#_Toc73114319)

[3.1 Проблемы, возникающие при реализации 17](#_Toc73114320)

[3.2 Реализация хранения сцены 18](#_Toc73114321)

[3.3 Реализация захвата событий октодерева 19](#_Toc73114322)

ВВЕДЕНИЕ

В современном мире все чаще необходимы возможности по сохранению и обработке какой-либо информации о трехмерном пространстве. Однако, сохранение трехмерного пространства требует очень большого объема памяти, а его обработка может иметь кубическую сложность, что является большой проблемой при его обработке как в реальном времени, так и нет.

Для решения данной проблемы имеются алгоритмы сжатия данных и оптимизации операций, основанные на специальном виде деревьев: октодеревьях. Октодеревья решают проблему компактного хранения и быстрой обработки пространства. С их помощью возможно многократно сократить объем потребляемой памяти, обеспечить удобный интерфейс работы с ними.

Алгоритмы и структуры данных, основанные на октодеревьях применяются во множестве перспективных технологий, например, в компьютерной графике или алгоритмах разрешения коллизий. Примером являются разреженное воксельное октодерево. С его помощью возможна как реализация неполигональной графики, так и технологий трассировки лучей, и их модификации: алгоритмы трассировки путей.

Целью работы является создание программного продукта, сохраняющего в себе информацию о трехмерном мире при помощи октодеревьев, для последующей интеграции в него агентов, направленных на моделирование выполнения каких-либо задач. Внедрение оптимизаций по памяти и производительности в октодеревья и проведение сравнения потребления памяти относительно неоптимизированных деревьями.

# 1 **Теоретические положения об использовании деревьев в приложениях компьютерной графики**

## 1.1 Проблемы, возникающие при разработке трехмерного видеодвижка

В настоящее время современные игры и графические симуляторы требуют все большей детализации и скорости обработки, максимально приближаясь к тому, как выглядит реальный мир и изменения в нем, поэтому вопрос обработки изображения требует новых быстрых способов работы с трехмерными изображениями.

Проблемы с большими детализированными уровнями возникают из-за количества полигонов с которыми приходится иметь дело. Рисование всех полигонов в каждом кадре неэффективно. Для увеличения скорости было бы неплохо визуализировать только те полигоны, которые находятся в поле зрения, а чтобы игра работала еще быстрее, исключить сканирование каждого полигона сцены для определения того, видим ли он.

Как можно определить, какие полигоны видимы, не сканируя их все в каждом кадре? Решение заключается в разделении трехмерной модели (представляющей уровень) на небольшие фрагменты (называемые узлами, nodes), содержащие несколько полигонов. Затем узлы заключаются в дерево, которое можно быстро просканировать для определения того, какие узлы видимы. Затем видимые узлы визуализируются.

Для определения того, какие узлы видимы, используем пирамиду видимого пространства. Теперь вместо того, чтобы сканировать тысячи полигонов, сканируется небольшой набор узлов, чтобы определить, что рисовать, это дает существенное улучшение процесса рисования[[1]](#footnote-1).

## 1.2 K-мерные деревья поиска

K-мерное дерево — это несбалансированное дерево поиска для хранения и упорядочивания точек в k-мерном пространстве .

Существуют однородные и неоднородные k-d деревья. У однородных k-d деревьев каждый узел хранит запись. При неоднородном варианте внутренние узлы содержат только ключи, листья содержат ссылки на записи[[2]](#footnote-2).

Таблица 1. Сложность операций в k-мерном дереве поиска

|  |  |  |
| --- | --- | --- |
|  | В среднем | В худшем случае |
| Расход памяти | O(*n*) | O(*n*) |
| Поиск | O(log *n*) | O(*n*) |
| Вставка | O(log *n*) | O(*n*) |
| Удаление | O(log *n*) | O(*n*) |

Дерево разбивает K-мерное пространство на меньшие части посредством прямых (при K = 2), плоскостей (при K = 3) или гиперплоскостей. При K = 1 рассечение происходит точкой и, фактически, является двоичным поиском. В результате структура данных разбивает многомерное пространство на множество иерархически вложенных и, возможно, пересекающихся, прямоугольников (для двумерного пространства). В случае трехмерного или многомерного пространства это будут прямоугольные параллелепипеды (кубоиды) или параллелотопы.

Алгоритмы поиска используют ограничивающие прямоугольники для принятия решения о необходимости поиска в дочерней вершине. Таким образом, большинство вершин никогда не затрагиваются в ходе поиска. Это свойство K-деревьев обусловливает их применимость для баз данных, где вершины могут выгружаться на диск по мере необходимости или в области компьютерной графики, где они отсекают проверку лишних объектов.

## 1.3 Октодеревья

Октодерево (восьмеричное дерево) - тип древовидной структуры данных, в которой у каждого внутреннего узла ровно восемь «потомков». Каждый узел (англ. node) в дереве октантов делит пространство на восемь новых октантов. В региональной точке октодерева узел сохраняет явную трёхмерную точку, которая является «центром» разделения пространства для этого узла. Данная точка определяет один из углов каждого из восьми дочерних пространств. Октодеревья не могут считаться k-мерными деревьями, поскольку k-мерные деревья разделяются вдоль размерности, а октодеревья разделяются вокруг точки. Кроме того, k-мерные деревья всегда являются двоичными, что неверно для октодеревьев.[[3]](#footnote-3)

Все поддеревья октодерева являются восьмеричными деревьями.

Октотомические деревья являются естественным распространением концепции квадродерева для представления трехмерного пространства. Подобно тому, как при построении квадродерева область разбивается на четыре части, при построении октодерева трехмерный объект подразделяется на восемь кубов (октантов). Если какой-либо из октантов является однородным, т. е. он располагается либо целиком внутри объекта, либо целиком снаружи, разбиение заканчивается. Иначе, если октант не является однородным, т. е. октант пересекается граничной поверхностью объекта, он разбивается далее на восемь подоктантов. Процесс разбиения завершается, когда все листовые узлы октодерева станут однородными, возможно с некоторой погрешностью.

На рисунке (1) изображена структура октодерева и принцип разбиения пространства на более малые части.

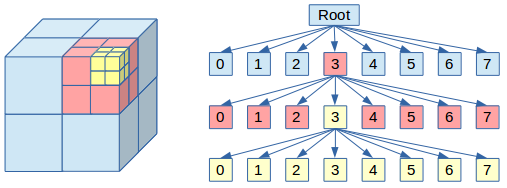


Рисунок 1 – принцип деления октодерева

Октодеревья формируются по следующему принципу: изначально создается корень дерева, имеющий размерность . При добавлении новых узлов пространство, описываемое узлом, дробится на 8 равных кубических частей до тех пор, пока не останется атомарный элемент размерности 1. Если узел уже использовался и раздроблен, дробления не происходит. Происходит выбор той части, в которой находится лист с искомыми координатами для вставки элемента. После достижения листа в него вставляются данные.

Пример вставки узла в октодерево с размерностью корня 8. Изначально дерево имеет лишь один корневой элемент, описывающий все пустое пространство. После чего в его узел добавляется один элемент. В этом случае, пространство разбивается на 8 частей, вычисляется часть, в которой находится требуемый лист и происходит рекурсивная вставка элемента в требуемую часть. Требуемая часть также является октодеревом. Данный процесс проиллюстрирован на рисунке 2. Узлы, в которых расположен символ «Е» являются пустыми листовыми узлами, а листья с цифрой – имеют некоторые данные, обозначаемые данным числом.

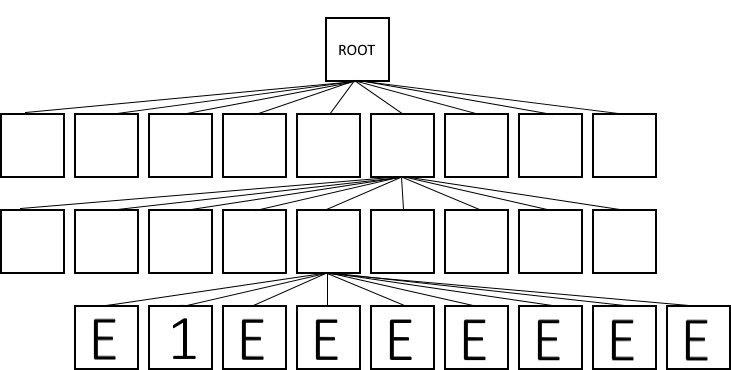


Рисунок 2 – добавление первого узла в октодерево

При попытке добавления еще одного элемента дальнейшего раздробления раздробленных элементов пространства уже не будет. Это продемонстрировано на рисунке 3 - в дерево был вставлен еще один элемент, но произошло раздробление лишь узла на предпоследнем уровне.

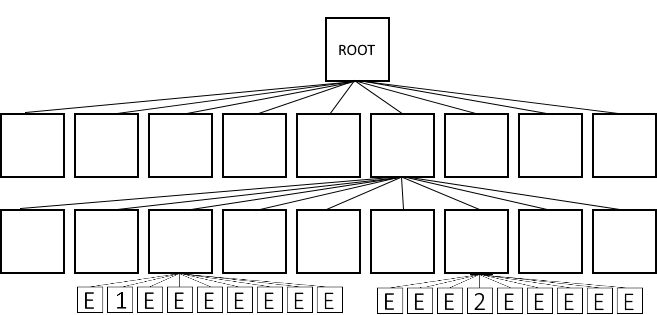


Рисунок 3 – дерево после добавление второго узла

В дальнейших главах будут рассмотрены способы оптимизации хранения и обработки трехмерного пространства посредствам октодеревьев.

Аналогично k-d деревьям, восьмеричные деревья бывают однородными и неоднородными по своей специфике – хранение и сжатие пространства, зачастую в этих задачах используются неоднородные деревья.

## 1.4. Наивная реализация октодерева

При наивной реализации алгоритмы и методы поиска отличаются от алгоритмов и методов поиска на обычных K-мерных деревьях поиска. Основными операциями интерфейса октодерева являются операции поиска, добавления узла, удаления узла, обхода дерева.

Дано октодерево T изначальной размерности , содержащее в себе n элементов.

Операция поиска узла *FIND(K)*, используемая для получения этих узлов, содержащих в себе некоторые данные, например, сведения о местоположении узла в пространстве. Так как дерево не является деревом поиска, при наивной реализации необходим обход всех узлов дерева и их проверка на содержание ключа K. Поэтому сложность операции поиска является линейной: O(n).

Операция получения узла с заданными координатами *GET(X, Y, Z)* не требует никаких обходов и представляет собой последовательность прямых переходов по узлам дерева, поэтому зависит лишь от размеров исходного дерева: O(m).

Операция добавления в дерево узла с заданными координатами или замена данных на заданных координатах *SET(X, Y, Z, K)* аналогична операции *GET*.

Операция удаления узла дерева REMOVE(X, Y, Z) аналогична операции *SET(X, Y, Z, K).*

# 2 Методы оптимизации алгоритмов на октодеревьях

## 2.1 Общая оптимизация октодерева

Одним из путей оптимизации дерева является добавление информации об нижестоящих листовых узлах во внутренние узлы дерева. Таким образом, ценой увеличения количества потребляемой памяти, появляется возможность быстро анализировать нижестоящие узлы. В операции *FIND(K)* появляется возможность преждевременно прервать поиск элемента в случае, если искомый узел исчез из списка нижестоящих узлов. Во время итерации по узлам дерева появляется возможность прекратить итерацию вглубь поддерева в случае, если там присутствуют или отсутствуют какие-либо данные[[4]](#footnote-4).

Однако, вышеописанная оптимизация значительно увеличивает количество потребляемой памяти. Имеет смысл объединять несколько одиночных соседних узлов, образующих собой куб со стороной  в один большой узел – таким образом значительно экономится память на удалении внутренних и внешних узлов при большом количестве однотипных данных.

Комбинируя вышеописанные оптимизации можно достичь значительного ускорения обработки и экономии памяти при сохранении информации об трехмерном пространстве в октодерево.

Во время сохранения или передачи графа добавленные избыточные данные можно убрать и добавлять автоматически во время приема данных, если в этом будет необходимость. В некоторых случаях имеет смысл передавать не все узлы октодерева, а только те, которые будут необходимы. Например, в воксельной графике не имеет смысла передавать невидимые воксели в графический процессор, так как они не влияют на вычисления сцены.

В результате вышеописанных оптимизаций несколько усложняется процесс построения графа.

Рассмотрим следующий пример графа G. Пустые листовые узлы помечены буквой «Е», узлы с данными – некоторой цифрой. Красными и зелеными узлы с данными, причем в красных и зеленых узлах находятся разные данные. На рисунке 4 граф уже имеет 14 заполненных листовых элементов.

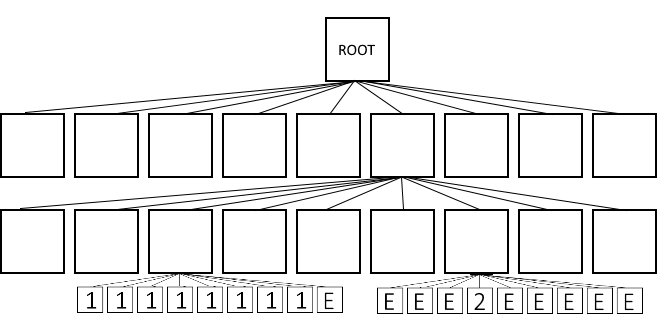


Рисунок 4 – граф G.

В октодерево вставляют новый элемент на место зеленого в левом узле. После этого все дочерние узлы левой вершины станут красными. В этом случае они будут объединены в один большой узел. На рисунке 5, в результате данных преобразований, узел второго уровня обозначается как узел, имеющий в дочерних только красные вершины. Теперь он хранит в себе лишь информацию о себе самом, а информация об дочерних вершинах не хранится. Таким образом выходит значительная экономия памяти в данном узле.

В реальных условиях похожие данные, зачастую, находятся в одном месте и объединяются с высокой вероятностью.

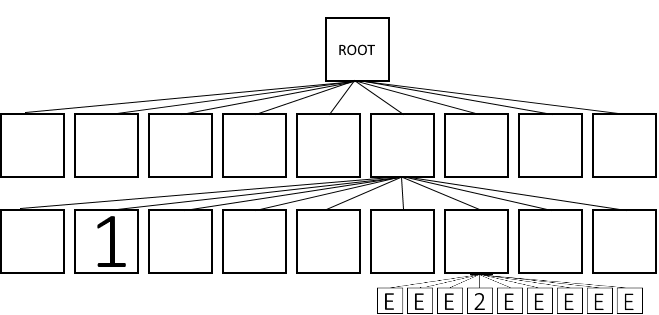


Рисунок 5 – граф G после добавления узла

## 2.2 Сравнение производительности и расхода памяти в наивной и оптимизированной реализации

Для анализа потребления памяти октодеревом и скорости выполнения операций на дереве возможно варьировать несколько переменных. А именно размерность дерева, количество возможных значений на его листьях, процент наполненности дерева. Во всех описанных ниже графиках используется сравнение оптимизированной версии октодерева и его наивной реализации.

Для каждого теста производилось 10 генераций случайных деревьев, после чего находились средние показатели по каждому из анализируемых параметров.

На рисунке 6 изображен график зависимости потребления оперативной памяти в зависимости от размеров октодерева. В каждом листовом узле могут храниться 32 различных значения, которые равномерно распределены по пространству. Синяя линия изображает наивную реализацию дерева, а красная - оптимизированную. Каждое дерево заполнено на 75%. Пунктирная линия обозначает неоптимизированную реализация, а сплошная – оптимизированную.

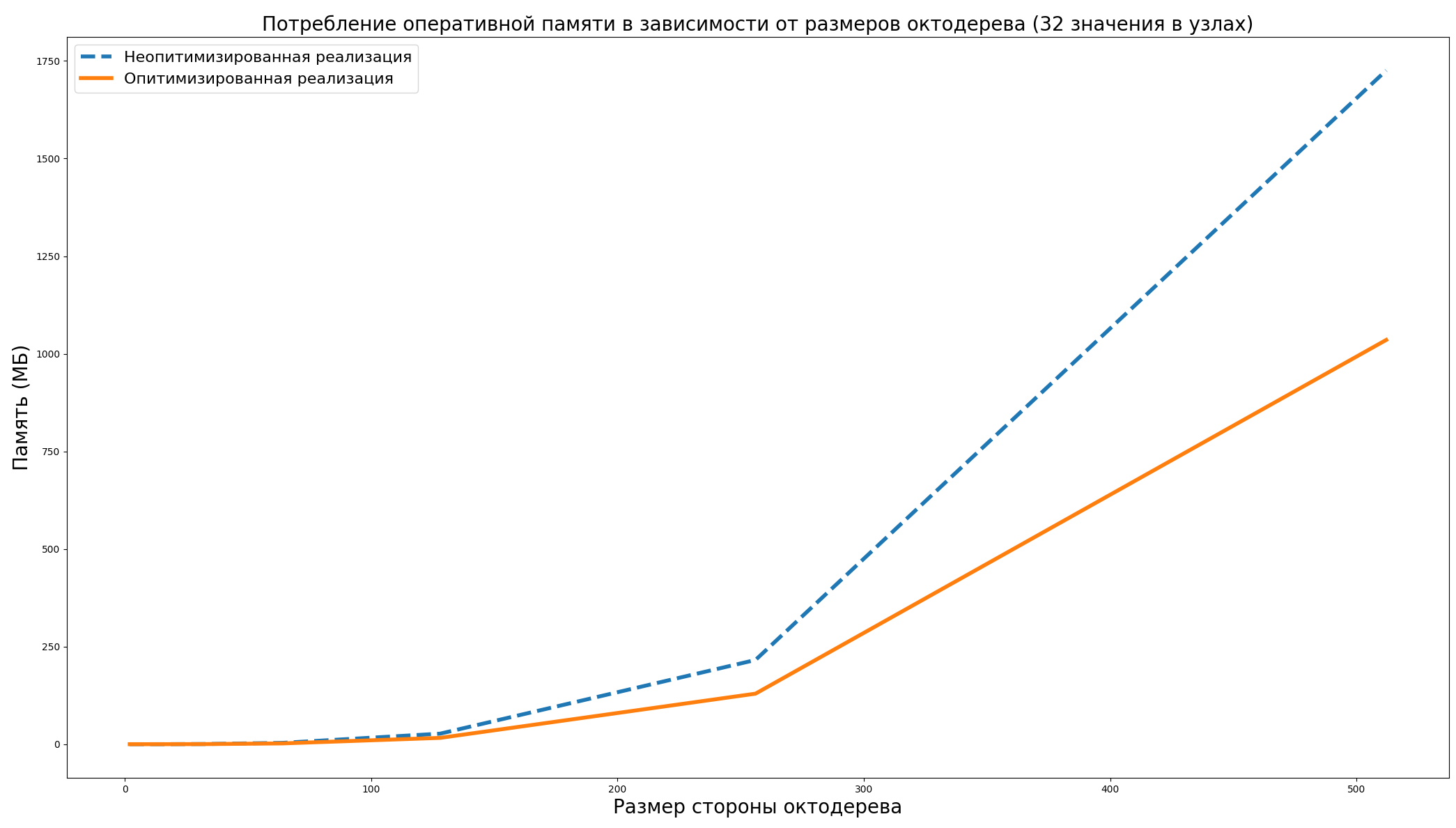


Рисунок 6 – график зависимости потребления оперативной памяти в зависимости от размеров октодерева.

На рисунке 7 изображен график зависимости потребления оперативной памяти потребления оперативной памяти в зависимости от размеров октодерева. В каждом листовом узле могут храниться 32 различных значения. Однако, если выше было совершенно случайное распределение данных в узлах, в данном случае они имеют некоторую зависимость. Данные в нижних, средних и верхних узлах являются однородными, относительно первого анализируемого дерева. Такое поведение моделирует использование октодерева в реальных условиях. Пунктирная линия обозначает неоптимизированную реализация, а сплошная – оптимизированную.

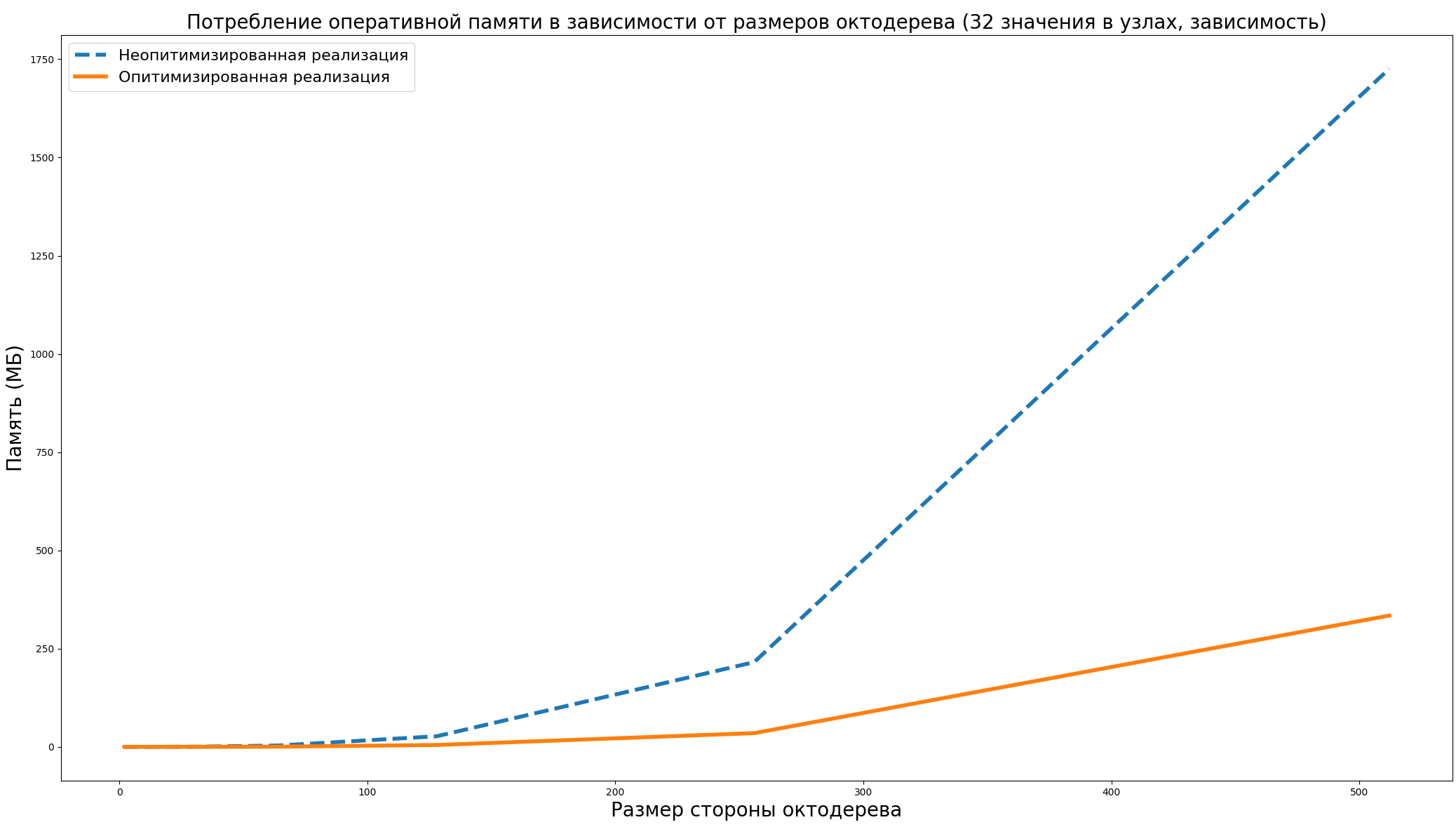


Рисунок 7 – график зависимости потребления оперативной памяти в зависимости от размеров октодерева.

По графикам можно заметить прирост производительности начиная с размерности дерева в 32. Это связано с тем, что в очень маленьких деревьях соотношение накладных расходов на сохранение метаданных по отношению к самим данным достаточно велико.

## 2.3 Упаковка октодерева в одномерный массив

Для отправки дерева по сети, или сохранения октодерева в ПЗУ используются алгоритмы упаковки, упаковывающие дерево в одномерный массив. Такое представление позволяет как экономить память, так и представить октодерево в виде, удобном для обработки на видеопроцессоре. На старых моделях видеопроцессоров такое представление является единственным доступным для обработки. Рассмотрим пример алгоритма, упаковывающего воксельное дерево для отправки его в GPU. Такие деревья называют Sparse Voxel Octree (SVO, Разреженное воксельное октодерево). Каждый воксель можно описать структурой, представленной на рисунке 6, сохраняющей позицию, цвет, нормаль и смещение:

struct Voxel

{

uint position;

uint color;

uint normal;

uint pad; // aligment

};

Рисунок 6 − представление вокселя в формате структуры (язык HLSL)

Каждый листовой узел октодерева имеет ссылку на объект данной структуры. Таким образом, данные октодерева ссылаются на внешнюю коллекцию вокселей, содержащее их описание.

Вышеописанная структура данных может быть легко представлена в плоском виде и сохранена в 2D-текстуре формата R32\_UINT. На рисунке 7 проиллюстрирована схематическая структура полученной текстуры:

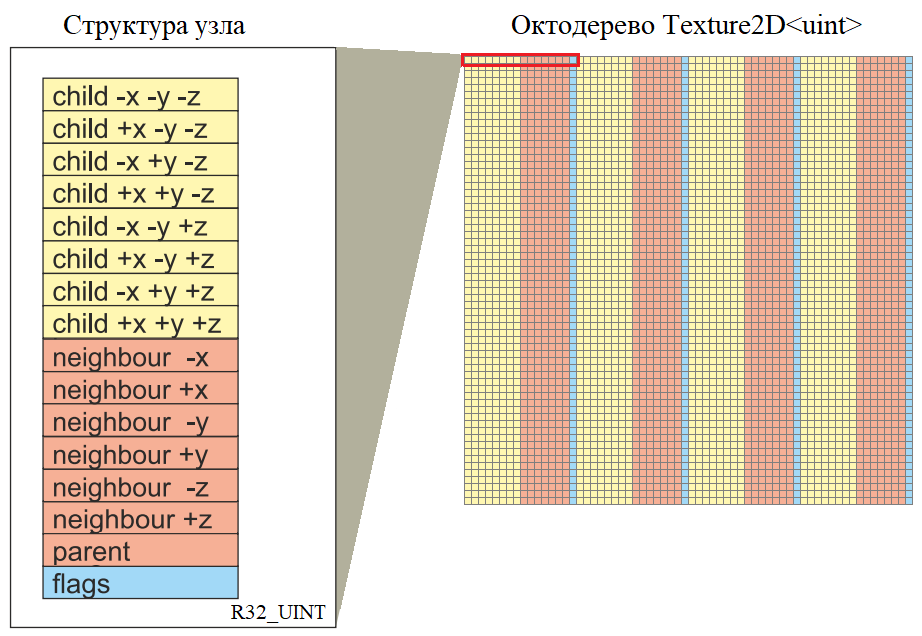


Рисунок 7 ­ Представление октодерева в формате 2D-текстуры

В листьях SVO на последнем уровне будут храниться индексы соответствующих вокселей из этого массива. При таком сжатии структуры хранятся исключительно воксели, которые хранят в себе полезную информацию. Пустых узлов не остается.

Конечно, имеется возможность построить октодерево обратно, исходя из массива вокселей[[5]](#footnote-5).

# 3. Реализация октодеревьев на практике

## 3.1 Проблемы, возникающие при реализации

При реализации октодеревьев для практического применения, зачастую, необходимо ввести дополнительные действия и операции во время добавления, удаления и обновления узлов, связанные с его практическим применением.

При реализации системы хранения и управления трехмерным миром, построенным на октодереве, одним из требований его функционирования была возможность его визуализации посредствам графических библиотек. Однако, создавать вызов отрисовки для каждого атомарного элемента мира является нецелесообразным: накладные расходы на вызов отрисовки небольшого фрагмента многократно более затратны, нежели один вызов для большого фрагмента. Вторым требованием данной системы, частично вытекающим из первого, является необходимость порождения и отлова событий, связанных с изменениями внутри дерева. Третьим – необходимость динамической загрузки и выгрузки сцены в оперативную память для каждого агента, присутствующего на ней

При реализации системы отлова событий возникают некоторые сложности. Во-первых, необходимо передавать информацию о событии наверх из дерева, сохраняя некоторое количество метаинформации. Во-вторых, нужно учитывать, что событие может быть нескольких типов и возвращать обобщенный класс события.

## 3.2 Реализация хранения сцены

Для решения проблемы динамической загрузки и выгрузки из оперативной памяти частей сцены разработана схема его разделения на несколько октодеревьев. Кроме того, такая схема позволяет обойти ограничение октодеревьев – они покрывают только кубическое пространствою. Большая сцена делится на более малые участки размером в  (ширина – высота – длина): куски (Chunk). Они, в свою очередь, поделены на октодеревья размерности 16, представленные классом Octree. Диаграмма классов, описывающая это представлена на рисунке 8.

Для экономии каждый узел октодерева разделяются на внутренние (InnerNode) и листовые (LeafNode). Помимо отличия логики в удалении, установке и иных действиях с данными, различаются типы хранимых данных. Листовые узлы хранят исключительно данные, а промежуточные – лишь ссылки на другие узлы: либо 8 листовых, либо 8 внутренних.

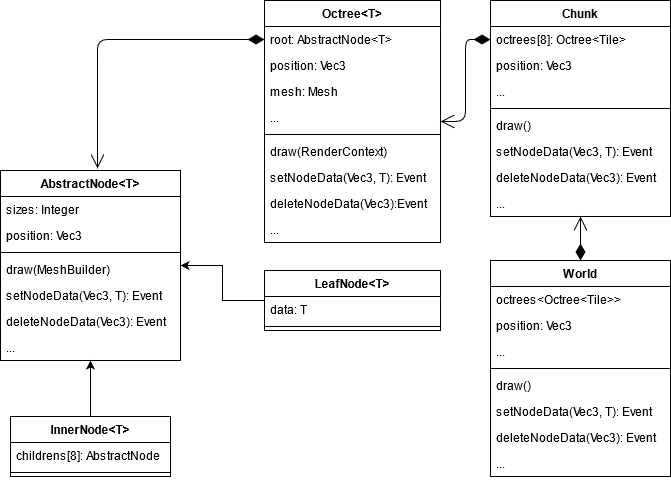


Рисунок 8 – фрагмент диаграммы классов хранения мира

При такой реализации становится возможная динамическая мира для каждого агента. Движок приложения может легко подгружать куски мира необходимые только каждому конкретному агенту на сцене. Определяется кусок, в котором расположен агент и происходит загрузка кусков мира вокруг него. Данный механизм особенно хорошо показывает себя в среде нескольких агентов, расположенных далеко друг от друга.

## 3.3 Реализация захвата событий октодерева

Одним из важных элементов практической реализации октодеревьев во многих прикладных приложениях является реализация системы событий. Необходимо передавать события об изменении, обновлении, удалении и иных действиях с элементами дерева.

Схема классов, используемых при реализации системы описаны в следующей диаграмме классов, изображенной на рисунке 9.

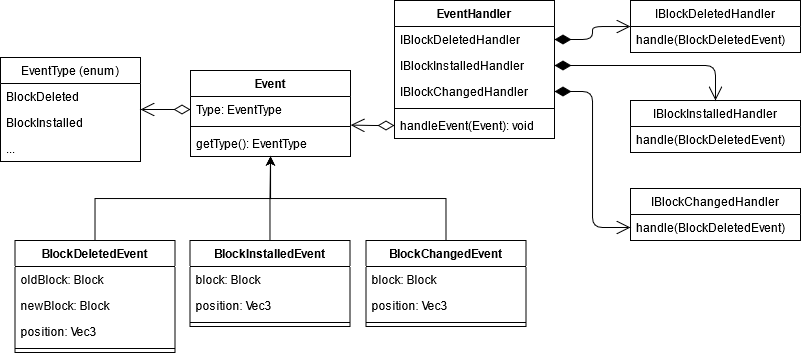


Рисунок 9 – фрагмент диаграммы классов регистрации событий

Исходя из рисунков 8 и 9 видна схема передачи событий. Событие генерируется в классе октодерева Octree и перебрасывается в класс куска пространства Chunk и соответственно в класс сцены World. После чего класс, который контролирует класс World может передать событие в обработчик событий EventHandler, который произведет обработку события. Для регистрации событий, их перехватывает класс EventHandler, который вызывает соответствующий метод обратного вызова. На диаграмме представлены не все возможные методы октодеревьев – лишь некоторые, большее количество событий создается абсолютно аналогично.

Некоторые из событий могут передаваться далее, в иные обработчики, помимо EventHandler, если то необходимо.

ЗАКЛЮЧЕНИЕ

В результате работы были исследованы октодеревья, их практическое применение. Были реализованы деревья для сохранения трехмерной сцены, исследованы способы оптимизации, возможные при их реализации, проведен сравнительный анализ потребления памяти деревьями до оптимизаций и после них.

Реализован трехмерный мир для размещения в него агентов, основанный на вышеописанных октодеревьях, путем их группирования в отдельные его части.

Рассмотрены дальнейшие перспективы разработки приложения и использования полученных во время работы данных. Намечен вектор работы.

СПИСОК ИСТОЧНИКОВ

1. Шикин Е. В., Боресков А. В. Компьютерная графика. Динамика, реалистические изображения. -М.: ДИАЛОГ–МИФИ, 1996. - 288 с. ISBN 5-86404-061-4

2. Сенюкова О. В. Сбалансированные деревья поиска:Учебно-методическое пособие. –М.: Издательский отдел факультета ВМиК МГУ имени М.В. Ломоносова (лицензия ИД N 05899 от 24.09.2001 г.); МАКС Пресс, 2014. –68 c. ISBN – 978-5-89407-528-0

3. K. Yamaguchi, "Octtree related data structures and algorithms", IEEE CG&A, vol. 4, no. 1, pp. 53-59, January 1984

4. M. Lovey, "A Hybrid Ray Traces for Rendering polygon and volume data", IEEE CG &A, vol. 10, no. 2, pp. 33-40, March 1990.

5. Lambert M. Surhone, Miriam T. Timpledon, Susan F. Marseken “Voxel: 3D Computer Graphics, Regular Grid, Raster Graphics, Coordinate System, Polygon, Volumetric Display, Polynomial Interpolation, Hounsfield Scale, Volumetric Flow Rate”, Betascript Publishing, 2010, ISBN – 6130321007, 9786130321000

1. Шикин Е. В., Боресков А. В. Компьютерная графика. Динамика, реалистические изображения. -М.: ДИАЛОГ–МИФИ, 1996. - 288 с. ISBN 5-86404-061-4 [↑](#footnote-ref-1)
2. Сенюкова О. В. Сбалансированные деревья поиска: Учебно-методическое пособие. –М.: Издательский отдел факультета ВМиК МГУ имени М.В. Ломоносова (лицензия ИД N 05899 от 24.09.2001 г.); МАКС Пресс, 2014. –68 c. ISBN – 978-5-89407-528-0 [↑](#footnote-ref-2)
3. K. Yamaguchi, "Octtree related data structures and algorithms", IEEE CG&A, vol. 4, no. 1, pp. 53-59, January 1984 [↑](#footnote-ref-3)
4. M. Lovey, "A Hybrid Ray Traces for Rendering polygon and volume data", IEEE CG &A, vol. 10, no. 2, pp. 33-40, March 1990. [↑](#footnote-ref-4)
5. Lambert M. Surhone, Miriam T. Timpledon, Susan F. Marseken “Voxel: 3D Computer Graphics, Regular Grid, Raster Graphics, Coordinate System, Polygon, Volumetric Display, Polynomial Interpolation, Hounsfield Scale, Volumetric Flow Rate”, Betascript Publishing, 2010, ISBN – 6130321007, 9786130321000 [↑](#footnote-ref-5)