|  |  |
| --- | --- |
| Пермский филиал федерального государственного автономного  образовательного учреждения высшего образования  «Национальный исследовательский университет  «Высшая школа экономики» | |
| *Факультет экономики, менеджмента и бизнес-информатики* | |
| Шевчук Михаил Романович | |
| **ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ ИГРЫ С РАЗРУШАЕМЫМ ЛАНДШАФТОМ** | |
| *Выпускная квалификационная работа* | |
| по направлению подготовки *09.03.04 Программная инженерия* образовательная программа «Программная инженерия» | |
| Рецензент | Научный руководитель старший преподаватель кафедры информационных  технологий в бизнесе Г. И. Рустамханова  Консультант старший преподаватель кафедры информационных  технологий в бизнесе В. В. Лебедев |
| Пермь, 2020 год | |

# Аннотация

Оглавление

[Аннотация 2](#_Toc59659908)

[Введение 4](#_Toc59659909)

[Глава 1. Анализ предметной области 7](#_Toc59659910)

[1.1. Анализ существующих игр 7](#_Toc59659911)

[1.2. Анализ существующих методов генерации деформируемого ландшафта 7](#_Toc59659912)

[1.3. Выявление требований к разрабатываемой игре 8](#_Toc59659913)

[Глава 2. Проектирование игровых компонентов и механик 9](#_Toc59659914)

[2.1. Конструирование архитектуры разрабатываемой игры 9](#_Toc59659915)

[2.2. Проектирование механики генерации TD 9](#_Toc59659916)

[2.3. Проектирование геймплея игрока 9](#_Toc59659917)

[2.4. Проектирование пользовательского интерфейса игры 9](#_Toc59659918)

[Глава 3. Реализация игровых компонентов и механик 10](#_Toc59659919)

[Глава 4. Тестирование игрового процесса 11](#_Toc59659920)

[Заключение 12](#_Toc59659921)

[Библиографический список 13](#_Toc59659922)

# Введение

Рынок компьютерных игр с каждым годом набирает новые обороты. Всё больше крупных компаний, таких как BMW, Intel, Nike, Xiaomi, инвестируют в киберспорт миллионы долларов. В период пандемии COVID-19 индустрия компьютерных игр стала одной из немногих отраслей, которая не только существенно не пострадала от карантинных ограничений, но и показала рост прибыли и количества клиентов. К примеру, в Китае выручка рынка видеоигр за первые месяцы 2020 года выросла на 25,2% по сравнению с концом 2019, а прибыль с мобильных игр увеличилась на 37,6%, что говорит о том, что индустрия в ближайшее время будет только расширяться [1].

Особенностью компьютерных игр является разнообразие жанров и механик игрового процесса. Одной из таких механик является механика «разрушаемого окружения» (destructible environment, далее DE). Под DE подразумеваются игровые объекты, которые могут быть разрушены или деформированы в результате действий игрока. Это могут быть строения, предметы интерьера, растительность и другие искусственные, либо природные структуры. Способность игрока разрушить стену или препятствие в лабиринте и таким образом создать себе новый вариант пути является примером механики DE.

Существует отдельный подвид DE – «деформируемый ландшафт» (terrain deformation, далее TD), имеющий отличия и сложности в плане реализации. Если при разрушении объекта, находящего на ландшафте, на программном уровне достаточно просто заменить его текстуру или удалить из памяти, то при деформации самого ландшафта меняется его геометрия. К примеру, деформация исходного плоского полигона поверхности приводит к разбиению его на множество полигонов, больше не лежащих в его плоскости. При этом постоянное динамическое изменение ландшафта создаёт большую нагрузку на аппаратные ресурсы компьютера. И по этой причине механика TD меньше всего используется в самом динамичном жанре игр – «шутер».

В данной работе представлена разработка и реализация игры в жанре «шутер» с использованием механики TD.

**Объектом исследования** является игра в жанре «шутер» с механикой деформации ландшафта, а **предметом исследования** является процесс генерации деформируемого ландшафта для игр жанра «шутер».

Цель выпускной квалификационной работы – разработка 3D игры в жанре «шутер» с разрушаемым ландшафтом. Для достижения поставленной цели нужно выполнить следующие задачи:

1. Проведение анализа предметной области:
   1. Анализ существующих игр, использующих механику TD.
   2. Анализ существующих методов генерации TD.
   3. Описание механик разрабатываемой игры.
   4. Выявление функциональных требований.
2. Проектирование игровых механик и игрового интерфейса:
   1. Проектирование механики генерации TD.
   2. Проектирование механики передвижения игрока.
   3. Проектирование механики взаимодействия игрока с окружением.
3. Реализация игровых механик и игрового интерфейса:
   1. Выбор игрового движка для реализации.
   2. Реализация игровых механик.
   3. Разработка игровой графики и пользовательского интерфейса.
4. Тестирование игровых механик.

Степень разработанности проблемы способствует выполнению поставленной цели, так как существуют актуальные исследования на тему динамически изменяемых ландшафтов, а также существуют примеры игр, использующих данную механику. Практической значимостью игры является реализация метода с использованием нового для проблемы инструментального средства, что может привести к расширению спектра игровых механик и жанров на рынке в случае успеха продукта. Теоретической значимостью является доказательство сходимости алгоритмов и методов при решении проблемы генерации TD для динамических игр жанра «шутер».

В процессе работы предполагается использование следующих методов исследования:

1. Анализ – рассмотрение существующих игр с подобными механиками и рассмотрение существующих способов реализации генерации TD.
2. Дискретизация – один из методов генерации TD – представление его не как одной поверхности, а в виде совокупности примитивов – вокселей.
3. Визуализация – рендеринг полученного результата на мониторе игроков.
4. Анализ предметной области

В первой части данной главы описан анализ существующих решений – игр, которые используют механику TD. Затем, на основе полученных данных выделяются основные существующие методы реализации механики TD, приводится анализ исследований этих методов и других, которые ранее возможно в играх не использовались. Затем, на примере разобранных игр, выделяются необходимые механики и функциональные и нефункциональные требования к игре, и требования дополняются новыми механиками для реализуемой игры.

Список исследовательских вопросов:

1. Какие существуют игры, использующие механику TD? В каком жанре они созданы и как именно используют TD? Какие методы генерации TD при этом применяются?
2. Что из себя представляют методы TD? Как они реализуются? Какие имеют плюсы и минусы?
3. Какие обязательные механики должна включать в себя игра? Какие дополнительные механики дали бы ей преимущество перед другими играми?

## 1.1. Анализ существующих игр

Для разработки 3D игры в жанре «шутер», использующей механику TD, требуется рассмотрение существующих решений с целью выделения:

1. используемых способов создания разрушаемого ландшафта;

2. общих паттернов жанра;

Проводить анализ существующих решений будем по следующим параметрам:

1. Степень разрушаемости окружения.

2. Способы взаимодействия игрока с окружением.

3. Положение игровой камеры.

4. Геймплей игрока.

5. Особенности игрового ландшафта.

### 1.1.1. Worms 3D

## 1.2. Анализ существующих методов генерации деформируемого ландшафта

Как мы можем видеть, игры, в которых игрок может так или иначе деформировать ландшафт, используют для этого в основном два подхода: изменение полигональной сетки ландшафта и генерация ландшафта с помощью вокселей. Будем сравнивать эти способы, основываясь на следующих критериях:

1. Как работают эти подходы, и как это может повлиять на геймплей?

2. По какому алгоритму эти подходы деформируют ландшафт и какие проблемы этот алгоритм может создать?

3. Каким образом эти методы могут быть оптимизированы и может ли эта оптимизация быть реализована в нашей работе?

### 1.2.1. Генерация ландшафта с помощью полигональной сетки

Чаще всего в игре нам не важна внутренность игрового объекта, игрок взаимодействует только с поверхностями. В таком случае используется тесселяция поверхности. Тесселяция – это процесс разбиения поверхности на набор многоугольников - полигонов [5]. Традиционно разработчики делят поверхности на треугольники, что называется триангуляцией, по следующим причинам:

* треугольник – простейший тип многоугольника;
* треугольник всегда плоский, так как его вершины эту плоскость и определяют;
* в результате практически любых преобразований треугольник останется треугольником;
* большинство современного оборудования создано на основе треугольной растеризации [4].

Таким образом треугольники становятся кусочно-линейной аппроксимацией поверхности (рис указать).

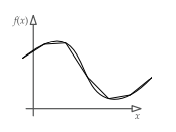


Рисунок .2. Демонстрация триангуляции как приближения к поверхности

Так как полигональная сетка является только «обволакивающей» поверхностью пустого трёхмерного пространства, это накладывает некоторые ограничения. Например, сильно затруднится генерация таких вещей как пещеры или мосты. Другой проблемой будет моделирование динамических объектов. Если нам нужно показать, например, волны на поверхности моря, то использование полигонов в этой ситуации потребует создание дополнительных объектов (брызги воды), изменение старых (деформированная волна), и так далее.

С тесселяцией неразрывно связано такое понятие как уровень детализации (level of detail, далее LOD). Фиксированная тесселяция может привести к тому, что на поверхности фигуры будут видны блоки, как на рисунке выше (тот же рисунок). Это решается увеличением количества полигонов, но, если каждый объект в игровом мире будет постоянно отрисовываться подробно, это сильно ударит по скорости компьютера, так как фигур на экране слишком много. Для решения этой проблемы подготавливают несколько уровней тесселяции поверхности объекта. Если камера близко к объекту – детализируем его максимально, если далеко – количество полигонов можно в разы уменьшить. На рисунке (рис ниже) левая поверхность разбита на 5000 треугольников, в то время как правая всего на 200 [4]. Таким образом, мы можем получить детализированное изображение, при этом сохранив большое количество памяти GPU.

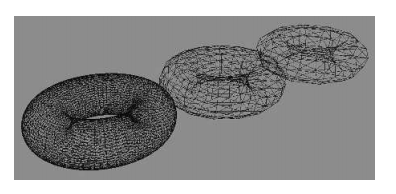


Рисунок .3. Три уровня детализации объекта

Ландшафт в современных играх генерируется уже с расчётом LOD’ов, которые меняются по мере передвижения игровой камеры. К примеру, одно из решений создания полигональной сетки для ландшафта имеет следующий алгоритм [9]:

1. Определяется позиция и направление камеры игрока. Во входные данные поступают все видимые регионы ландшафта.

2. Запускается цикл, одна итерация соответствует одному региону. Если регион не попадает в поле зрение камеры, то он пропускается.

3. Затем проходит проверка, нужно ли детализировать регион. Чем ближе регион к игроку, тем больше он должен быть детализирован. Детализация проводится разбиением региона на 4 части. Затем для каждой части рекурсивно проходит такая же проверка. Выходными данными является детализированный по разной степени LOD’ов регион.

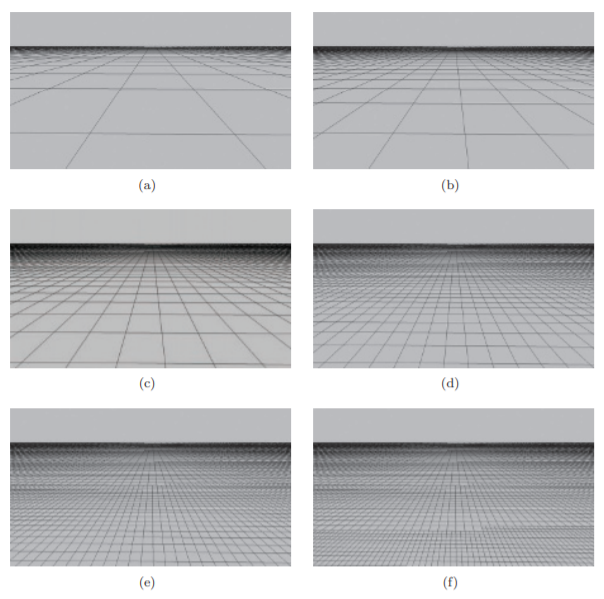


Рисунок 1.4. Вид полигональной сетки ландшафта на итерациях алгоритма N, N+1, N+2, N+3, N+4 и N+5 соответственно

Полигональный ландшафт, деформированный каким-либо воздействием, изменяется подобным образом. В месте деформации происходит детализация сетки, и, в зависимости от силы деформирующего воздействия, разные LOD’ы меняют свои координаты, и полигоны в данном месте перерисовываются. На рисунке ниже видно, как в местах, деформированных грузовым транспортом, LOD’ы намного больше, чем на ровной поверхности [10]. Однако такой метод требует много расчётов, и может замедлить процесс рендеринга изображения.

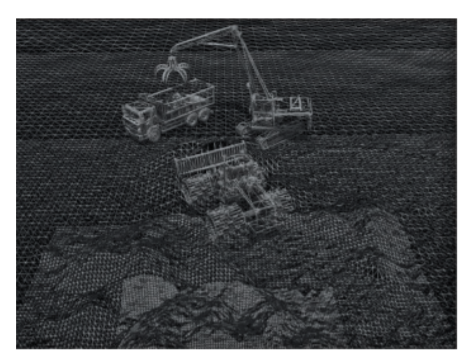


Рисунок 1.5. Уровень детализации ландшафта в местах деформации

После проведения тесселяции, координаты вершин треугольников объекта записываются в массив. Так как одна вершина может принадлежать сразу нескольким треугольникам, обычно разделяют массивы вершин и массивы индексов, комбинации которых уже и составляют треугольник. Делается это чтобы не отрисовывать и не освещать одну и ту же вершину несколько раз. Есть и другие структуры данных, направленные на экономию памяти компьютера, такие как «полоса» (Strips) и «веер» (Fans), использующие свои закономерности порядка записи индексов. Такой порядок также способствует удобному доступу к памяти для GPU [4]. Например, в структуре данных Strips, две вершины одного треугольника являются и двумя вершинами соседнего треугольника, записанного в массиве. Такой подход позволяет сильно сократить размерность массива.

Таким образом, полигональная сетка позволит генерировать хорошо детализированный ландшафт, а учитывая, что полигоны хранятся на GPU, процесс генерации будет происходить быстро. Однако, при деформации ландшафта, полигоны требуют слишком большого количества расчётов. Если в игре будет происходить слишком много событий, например, сразу несколько игроков будут стрелять по ландшафту, это может вызвать лаги и задержки.

### 1.2.2. Генерация ландшафта с помощью вокселей

Если пиксель является «элементом изображения» (picture element), то воксел является «элементом объёма» (volume element). Воксели можно сравнить с атомами, из которых состоит трёхмерное пространство, и их совокупность представляется в виде трёхмерной сетки. Каждый воксел является отдельным объектом и поэтому может хранить в себе необходимую для задачи информацию, такую как цвет, прозрачность (для симуляции таких вещей как дым, вода), плотность и тому подобное. Информация о местоположении вокселя не требуется, так как это определяется его индексом в сгенерированной сетке [5].

Основное отличие между полигонами и вокселями в том, что в то время, как полигоны покрывают пустое трёхмерное пространство, создавая полую фигуру, вокселями составляют объём этой фигуры. Другими словами, объём фигуры равен сумме вокселей внутри неё. Благодаря этому, решение проблемы деформации какого-либо объекта, состоящего из вокселей, сводится к задаче прибавления и вычитания этих вокселей в нужных местах [5]. Такая гибкость вокселей даёт им большую вариативность в использовании, чем у полигонов. Для нашей игры это значит, что мы сможем создавать деформацию любых масштабов, от глобального изменения формы объекта до полного его уничтожения.

Проблемой вокселей является больше потребление памяти, имеющую величину . То есть локация, размерностью в 1000 вокселей по каждой оси уже имеет размер в 1 000 000 000 блоков, что создаёт огромную нагрузку на память процессора. Однако, соседние воксели часто имеют схожие значения, а некоторые части сетки вообще могут быть пустыми. Поэтому одним из основных методов оптимизации воксельной сетки и построения воксельного мира является построение разряжённого воксельного октодерева (Sparse voxel octree, далее SVO). Такой подход использовали авторы игры Teardown, описанной выше [6], и такой подход использует физический движок PhysX, в данный момент принадлежащий компании NVIDIA [7].

Суть SVO состоит в следующем. За условный объект возьмём дерево, состоящее из ствола, какого-то количества веток и листьев. Окружим наш объект идентичной по размеру областью в форме куба, и начнём делить эти область на части. Каждая часть может быть рекурсивно детализирована в дальнейшем. Проводим эту детализацию до тех пор, пока каждой области не соответствует часть нашего объекта, имеющая один набор свойств, одинаковый для каждой точки в этой области. То есть в нашем примере одной области может соответствовать либо весь ствол, либо одна из веток, либо лист. Таким образом, получится граф или дерево областей, которые содержат в себе части нашего объекта. Эти области и являются наборами вокселей, характеризующими конкретную часть объекта. На рисунке ниже (указать рисунок) показано, как мы проделываем процедуру до тех пор, пока конечная область не будет содержать полностью один элемент или будет пустой. Первоначальный куб является корнем дерева, конечные области-воксели являются листьями. Если же мы как-либо изменим часть объекта, то соответствующая деформируемой части область детализируется далее. То есть пока объект не тронут, он представляет собой один большой «куб», который меньше нагружает систему. Такой подход позволит сильно сэкономить память процесса, превращая миллиард блоков в несколько сотен или тысяч (в зависимости от объекта).

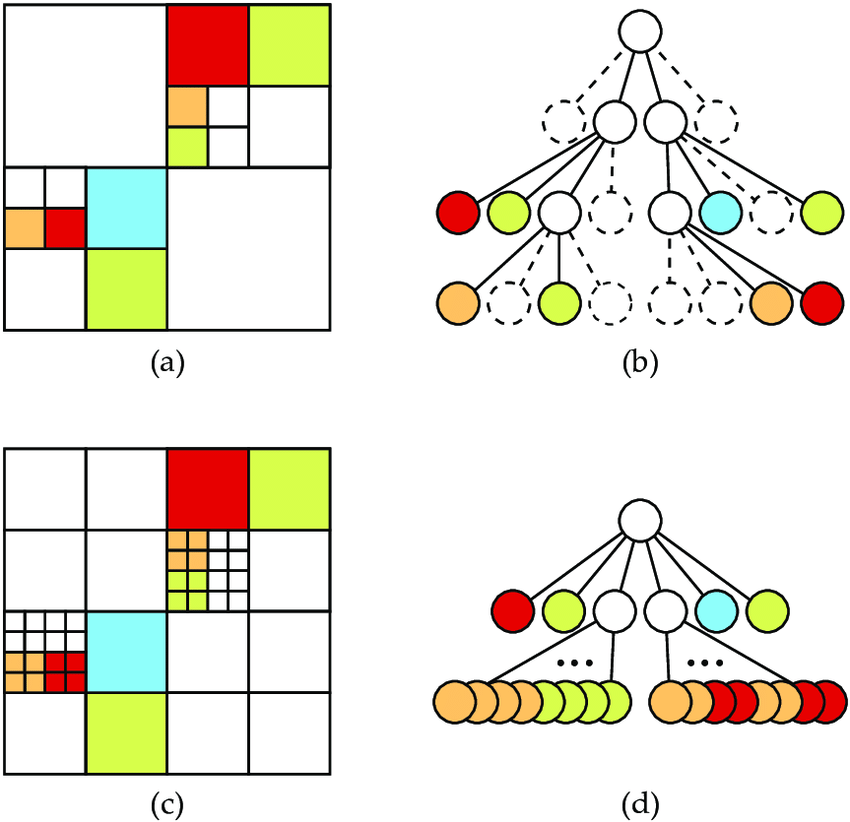


Рисунок 0.1 Примеры построения октодерева [8]

Для создания воксельного объекта используются вокселезация -преобразование объекта, состоящего из полигонов, в набор вокселей [13]. При создании воксельной сетки для каждого вокселя происходит проверка: лежит он внутри объекта или снаружи, затем эта информация записывается в воксель. Современные методы вокселезации позволяют генерировать воксельные объекты уже с построенным SVO [15].

Таким образом, процедура создания и деформирования воксельного объекта следующая:

1. Создание объекта в игровом пространстве, определение его мировых координат.

2. Вокселезация объекта. Воксели, находящиеся внутри объекта (формирующие его объём), содержат соответствующую информацию.

3. При событии, как-либо деформирующем объект, проводим следующий алгоритм:

a. Определяется позиция и масштаб деформации.

b. Пока соответствующий лист SVO больше масштаба деформации – детализируем данную ветвь SVO.

c. Когда лист, соответствующий масштабу деформации получен, проводим с этим набором вокселей соответствующую операцию: 1) если причиной деформации было разрушение объекта, то принадлежность к объекту у соответствующего набора удаляется; 2) если причиной деформации было увеличение объекта, то соответствующему набору вокселей присваивается принадлежность к объекту.

Как мы можем видеть, воксельный подход обладает большими затратами памяти, а также углы кубов, из которых состоят ландшафт и объекты, скорее всего будут видны. Однако, алгоритм построения SVO сильно сэкономит память компьютера, а специфичное изображение может свести на игровой стиль. Самым существенным плюсом воксельного подхода становится простота работы с деформацией ландшафта, являющаяся лишь вопросом удаления или добавления вокселей в нужном месте. Такая эффективность позволит быстро обрабатывать все события деформации и позволит сразу нескольким игрокам принимать участие в игре.

Составим таблицу сравнения для обоих подходов:

|  |  |  |
| --- | --- | --- |
| **Критерий** | **Полигональная сетка** | **Воксели** |
| 1. Влияние на геймплей | Внутренность объекта не важна | Составляют полный объём объекта |
| 2. Аппаратная часть | GPU | CPU |
| 3. Как деформирует ландшафт? | Сетка детализируется в месте деформации, для каждого треугольника рассчитывается новая позиция | Добавление и удаление вокселей в месте деформации |
| 4. Основная проблема | Большое количество расчетов при деформации | Нагрузка на память при большом разрешении игрового мира |
| 5. Насколько эффективны методы оптимизации | вместо со структурами данных | Зависит от вокселезируемого объекта, в приведенных выше примерах в 3.5 и 5.5 раз с SVO |

Для данной работу выбран метод вокселей с применением SVO. Решающими критериями являются простота процесса деформации объектов и более эффективный метод оптимизации. Такие плюсы позволят иметь в игре высокую производительность и позволит нескольким игрокам одновременно изменять игровой ландшафт.

## 1.3. Выявление требований к разрабатываемой игре

Результатами данной главы являются:

1. Выделение общих (необходимых) механик для игры на основе просмотренных аналогов.
2. Добавление дополнительных механик, отличных от разобранных в аналогах. При этом важно, чтобы дополнительные механики вписывались в концепцию выбранного метода генерации TD.
3. Определение списка функциональных требований на основе определённых механик.
4. Определение списка нефункциональных требований на основе определённых механик.
5. Проектирование игровых компонентов и механик

Список исследовательских вопросов:

1. Из каких подсистем состоит разрабатываемая игра? Какова связь между ними? Как с этими подсистемами взаимодействует пользователь?
2. Как организована механика генерации TD? Как организован геймплей игрока, его передвижение по карте, взаимодействие с игровыми объектами и ландшафтом?

## 2.1. Конструирование архитектуры разрабатываемой игры

Результатом данной главы является:

1. Составление списка игровых подсистем
2. Построение схемы взаимодействий этих подсистем

Результат достигается путём декомпозиции основных игровых механик, выявления связей и отношений между ними и построения диаграммы последовательностей и диаграммы понятий.

## 2.2. Проектирование механики генерации TD

Результатом данной главы является модель генерации и поведения ландшафта в зависимости от действий игрока. Данный результат достигается с помощью построения модели структуры ландшафта и рассмотрения возможностей изменения ландшафта в зависимости от выбранного метода генерации.

## 2.3. Проектирование геймплея игрока

Результатом данной главы является модель возможного поведения игрока и взаимодействия его с игровыми объектами.

## 2.4. Проектирование пользовательского интерфейса игры

Результатом данной главы является описание интерфейса игры, описание главного меню и настроек, управления игрока, настройка игровой камеры.

1. Реализация игровых компонентов и механик

Список исследовательских вопросов: какой инструмент (игровой движок) выбирается для разработки игры? Как реализовывается алгоритм генерации деформируемого ландшафта? Как реализовывается геймплей игрока?

1. Тестирование игрового процесса

Список исследовательских вопросов: какие существуют баги в игре и насколько они критичны? Корректно ли приложение отвечает на действия пользователя?

# Заключение

# Библиографический список

1. Как растёт индустрия компьютерных игр и киберспорт в условиях пандемии [Электронный ресурс]. – Режим доступа: <https://www.kommersant.ru/doc/4325738> (дата обращения 06.12.2020).
2. Zhao, J. Physically based modeling and animation of landslides with MPM // The Visual Computer. – 2019. – Vol. 35. – P. 1223-1235.
3. Spuy, van der R. Advanced Game Design with Flash – NY.:Apress, 2010 – P. 305-366.
4. Gregory, J. Game Engine Architecture – 3rd ed. – FL.:CRC Press, 2018. – P. 525-544.
5. Akenine-Moller, T. Real-time Rendering – 4th ed. – FL.:CRC Press, 2018. – P.545-589.
6. Разрушаемость в Teardown: как совместить геймплей и технологии [Электронный ресурс]. – Режим доступа: <https://media-xyz.com/ru/articles/714-razrushaemost-v-teardown-kak-sovmestit-geimple> (дата обращения 22.01.2021).
7. NVIDIA Blast [Электронный ресурс]. – Режим доступа: <https://developer.nvidia.com/blast> (дата обращения 22.01.2021).
8. Pätzold, M. Graphical model and simulation for THz-imaging: дис. канд. полит. наук. – Siegein, 2017. – С. 9.
9. Mistal, B. GPU Terrain Subdivision and Tessellation // GPU Pro 360. – 2018. – P. 177-194.
10. Pangerl, D. Dynamic GPU Terrain // GPU Pro 360. – 2018. – P. 223-237.
11. Kotfis, D. Octree Mapping from a Depth Camera // GPU Pro 7. – 2016. – P. 257-273.
12. Palmer, G. Physics for Game Programmers – NY.:Apress, 2018. – P. 83-138.
13. Schawaz, H., Seidel, P. Fast Parallel Surface and Solid Voxelization on GPUs // ACM Transations on Graphics. – 2010. – Vol.29. – P.179:1-179:10.
14. Kotfis, D. Cozzi, P. Octree Mapping from a Depth Camera // GPU Pro 7. – 2016. – P.257-273.
15. Baert, J. Lagae, A. Dutre, Ph. Out-of-Core Counstruction of Sparse Voxel Octrees // Computer Graphics Forum. – 2013. – Vol. 33. – P.220-227.