|  |  |  |
| --- | --- | --- |
| 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

Выпускная квалификационная работа «Проектирование и разработка игры с разрушаемым ландшафтом» подготовлена студентом НИУ ВШЭ – Пермь Шевчуком Михаилом. В работе рассматриваются теоретические и практические вопросы, касающиеся подхода к деформированию ландшафта в играх и разработки 3D игры в жанре шутер с механикой деформации ландшафта. В первой главе приводится анализ использования механики деформации ландшафта в играх и анализ актуальных исследований на эту тему. Во второй главе описывается проектирование разрабатываемой игры и её основных геймплейных механик. Далее описывается разработка и тестирование приложения.

Работа содержит \_ страниц формата А4 основного текста, включая 4 главы: анализ использования механики деформации ландшафта в играх, проектирование игровых компонентов и механик, их реализация и тестирование.

В основной части содержится \_ рисунков и \_ диаграмм, описывающих процессы, выполняемые системой.

Библиографический список состоит из \_ источников.

Оглавление

[Введение 4](#_Toc64732382)

[Глава 1. Анализ использования механики деформации ландшафта в играх 7](#_Toc64732383)

[1.1. Анализ существующих игр в жанре шутер с механикой деформации ландшафта 7](#_Toc64732384)

[1.1.1. Worms 3D (2003) 8](#_Toc64732385)

[1.1.2. Scorched 3D (2001) 9](#_Toc64732386)

[1.1.3. Battlefield 3 (2011) 10](#_Toc64732387)

[1.1.4. Сравнительная характеристика игр с механикой деформации ландшафта 11](#_Toc64732388)

[1.2. Анализ существующих методов генерации деформируемого ландшафта 12](#_Toc64732389)

[1.2.1. Генерация ландшафта с помощью полигональной сетки 12](#_Toc64732390)

[1.2.2. Генерация ландшафта с помощью вокселей 16](#_Toc64732391)

[1.3. Выявление требований к разрабатываемой игре 20](#_Toc64732392)

[1.3.1. Описание функциональных и нефункциональных требований к игре 21](#_Toc64732393)

[Глава 2. Проектирование игровых компонентов и механик 23](#_Toc64732394)

[2.1. Конструирование архитектуры разрабатываемой игры 23](#_Toc64732395)

[2.2. Проектирование механики генерации TD 23](#_Toc64732396)

[2.3. Проектирование геймплея игрока 23](#_Toc64732397)

[2.4. Проектирование пользовательского интерфейса игры 23](#_Toc64732398)

[Глава 3. Реализация игровых компонентов и механик 24](#_Toc64732399)

[Глава 4. Тестирование игрового процесса 25](#_Toc64732400)

[Заключение 26](#_Toc64732401)

[Библиографический список 27](#_Toc64732402)

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 [].

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 алгоритм генерации деформируемого ландшафта для игр жанра «шутер».

**Цель** выпускной квалификационной работы – разработка исследовательского прототипа 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. Сравнение по аналогам и требованиям – 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.

# Анализ использования механики деформации ландшафта в играх

В первой главе рассматривается следующий 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?

Вопросы первого блока поднимаются с целью выявления основных паттернов шутер жанра, выявления требований к разрабатываемому прототипу и рассмотрения способов создания динамического ландшафта. Затем проводится сравнительный анализ выявленных методов на предмет их релевантности по отношению к жанру шутер. Наконец, приводится список функциональных и нефункциональных требований, составленный на основе рассмотрения основных механик игр жанра шутер и дополнения их новыми, согласующимися с выбранным методом деформации ландшафта.

## 1.1. Анализ существующих игр в жанре шутер с механикой деформации ландшафта

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

* используемых способов создания разрушаемого ландшафта;
* общих паттернов жанра;

Проводить анализ существующих решений будем по следующим параметрам:

* Степень разрушаемости окружения.
* Способы взаимодействия игрока с окружением.
* Положение игровой камеры.
* Геймплей игрока.
* Особенности игрового ландшафта.

Существует сравнительно небольшое количество игр с DE, а тем более в жанре шутер []. Для анализа были выбраны 3 игры на основе их популярности и влияния на жанр в целом.

### 1.1.1. Worms 3D (2003)

Первые игры серии Worms уже поддерживали TD, и, бесспорно, дали толчок к развитию жанра шутер и механики TD в целом. В 2003 году выходит первая трёхмерная игра серии, которая не только сохранила предыдущие особенности, но и внесла новые. Переход в трёхмерное пространство дал игроку больше свободы и возможностей.

Игрок должен создать свою команду персонажей – червей, и, используя различное оружие, уничтожить команду противника. Персонажи ходят по очереди, сначала один червь из одной команды, затем следующий из другой, и так далее. Таким образом, за ход передвигается и стреляет только один персонаж, что позволит не нагружать систему большим количеством событий. Выбрав позицию, игрок выбирает оружие, которым хочет сделать выстрел, прицеливается, и выпускает снаряд. Каждый снаряд имеет свои особенности баллистики и поведения, что задаёт тактический аспект игры, а также свой наносимый урон. Цель игры – уничтожить всю команду соперника.

При движении игрока по карте, игровая камера находится в режиме «от третьего лица» для того, чтобы полностью было видно местность вокруг. При прицеливании камера переходит в режим «от первого лица» для удобства задания траектории полёта снаряда. Некоторые снаряды имеют свою камеру, например вид на карту сверху (см. рис 1.1).



Рисунок .. Два режима камеры в игре Worms 3D. Камера от третьего лица при передвижении слева и камера от первого лица при стрельбе справа

Игровой ландшафт разрушается от взрывов, вызванных выпущенными снарядами. Игроку доступен любой уровень разрушения, от рытья таким образом туннелей и пещер, до полного разрушения отдельных участков местности. Чтобы игрок не уничтожил всю карту, в игру добавлена вода, находящаяся в самом низу игровой карты и поднимающаяся с каждым ходом. При попадании в воду персонаж погибает. Игровая карта может иметь любую форму, имеются уровни как в виде обычных островов, так и средневековых замков. Такая мобильность создана при помощи использования вокселей. Воксели составляют игровой ландшафт как кирпичики, и при взрыве их часть удаляется, тем самым деформируя местность.

### 1.1.2. Scorched 3D (2001)

Другой «классикой» является игра Scorched 3D, которая является пошаговой артиллерийской стратегией. Игрок управляет танком, который размещается на игровой карте случайным образом. За один ход игрок может либо сделать выстрел, либо изменить позицию. Цель – уничтожить танки противников. При выстреле игрок выбирает угол вращения и силу выстрела, выбирает боеприпасы и инвентарь.

Игра имеет некоторые отличия от Worms 3D. Во-первых, игрок может видеть сразу обе камеры. Главная демонстрирует вид на карту сверху, её можно перемещать и рассматривать каждый кусок местности. Камера сбоку находится в режиме «от первого лица», и показывает, куда смотрит дуло танка (см. рис. 1.2).



Рисунок .. Игровой интерфейс игры Scorched 3D

Вторым отличием является модель разрушения ландшафта. В игре реализован динамический ландшафт через полигоны []. Это значит, что от взрывов ландшафт как бы прогибается, но при этом невозможно создание различных подповерхностных структур, таких как туннели и пещеры. Однако, это не исключает возможность полного уничтожения игровой карты.

### 1.1.3. Battlefield 3 (2011)

В отличие от предыдущих игр, Battlefield является более динамичной игрой, так как действие происходит в реальном времени. Игра является шутером от первого лица, и в неё можно играть как одному, проходя сюжет, так и в многопользовательском режиме. Проходя сюжет, требуется выполнить определённое задание, и уничтожение всех противников не является главной целью. В многопользовательском режиме игроки также действуют одновременно, играя за разные классы персонажей.

Battlefield 3 предоставляет большой уровень разрушаемости зданий, природных объектов и тому подобного. Использовался специальный физический движок Havok Physics, а создатели утверждали, что с его помощью возможно моделирование разрушение модели небоскрёба Бурдж-Халифа []. Однако уровень деформации ландшафта довольно поверхностный – в игре возможно создание кратеров от взрывов, но не более (см. рис. 1.3). Метод реализован с помощью генерации полигональной карты высот []. Однако, такой подход задал игре высокий уровень графики и детализации.



Рисунок .. Демонстрация разрушения ландшафта в игре Battlefield 3 [4]

### 1.1.4. Сравнительная характеристика игр с механикой деформации ландшафта

По итогам анализа вышеперечисленных игр составлена следующая таблица (табл. 1.1):

**Таблица 1.1. Сравнительная таблица существующих игр с механикой TD**

| **Критерий** | **Worms 3D** | **Scorched 3D** | **Battlefield 3** |
| --- | --- | --- | --- |
| Игровой темп | Пошаговая стратегия | Пошаговая стратегия | Шутер в реальном времени |
| Камера игрока | Переключение между камерами от первого и третьего лица | Камера от третьего лица с ограниченной камерой от первого | Камера от первого лица |
| Масштабы разрушаемости | Полное свобода разрушения | Практически полная разрушаемость с ограничениями | Практически полная разрушаемость зданий, поверхностная разрушаемость ландшафта |
| Мультиплеерный режим | Присутствует, но игроки ходят по очереди | Присутствует, но игроки ходят по очереди | Присутствует, игроки действуют в реальном времени |
| Способ реализации TD | Воксели | Полигональная сетка | Полигональная сетка |

Как мы видим, существующие игры используют TD по-разному. Однако не были найдены решения, которые бы совмещали многогранность, тактический аспект этой механики и командную игру в реальном времени. Worms использует TD в геймплейный целях, позволяя разрушать ландшафт под игроками, чтобы уничтожить их, или рыть туннели, но является пошаговой игрой. В Battlefield же играет роль разрушение окружения в виде построек, а не конкретно ландшафта.

## 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 []. 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 [].

Таким образом треугольники становятся кусочно-линейной аппроксимацией поверхности (рис 1.4).

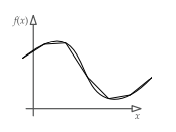


Рисунок .. Демонстрация триангуляции как приближения к поверхности []

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 (рис. 1.4). 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 (см. рис. 1.5) левая поверхность разбита на 5000 треугольников, в то время как правая всего на 200 []. Таким образом, мы можем получить детализированное изображение, при этом сохранив большое количество памяти GPU.

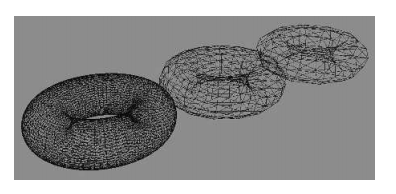


Рисунок .. Три уровня детализации объекта []

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 []:

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 (см. рис. 1.6).

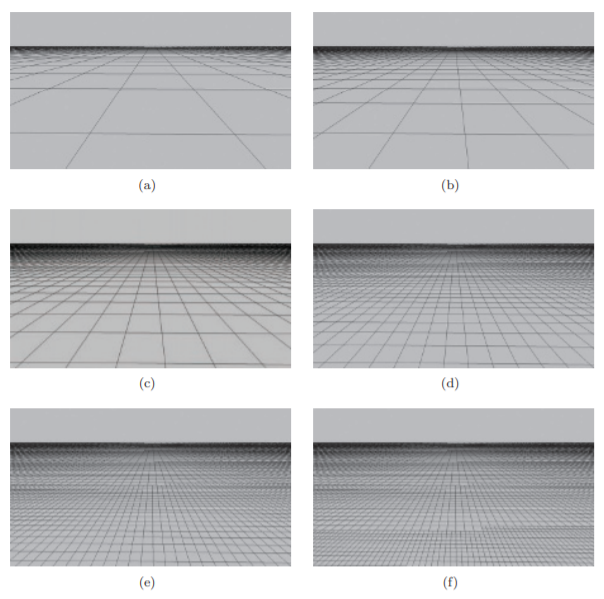


Рисунок .. Вид полигональной сетки ландшафта на итерациях алгоритма N, N+1, N+2, N+3, N+4 и N+5 соответственно []

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 (см. рис 1.7) ниже видно, как в местах, деформированных грузовым транспортом, LOD’ы намного больше, чем на ровной поверхности []. Однако такой метод требует много расчётов, и может замедлить процесс рендеринга изображения.

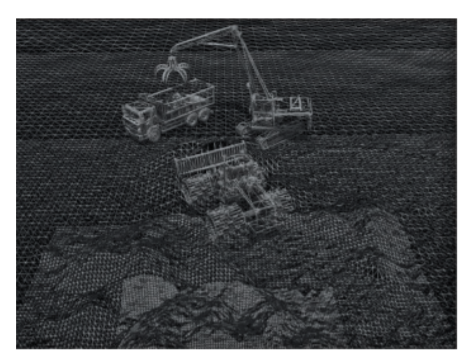


Рисунок .. Уровень детализации ландшафта в местах деформации []

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 []. 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 [].

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 []. 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 [], и такой подход использует физический движок PhysX, в данный момент принадлежащий компании NVIDIA [].

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 (рис. 1.8) 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).

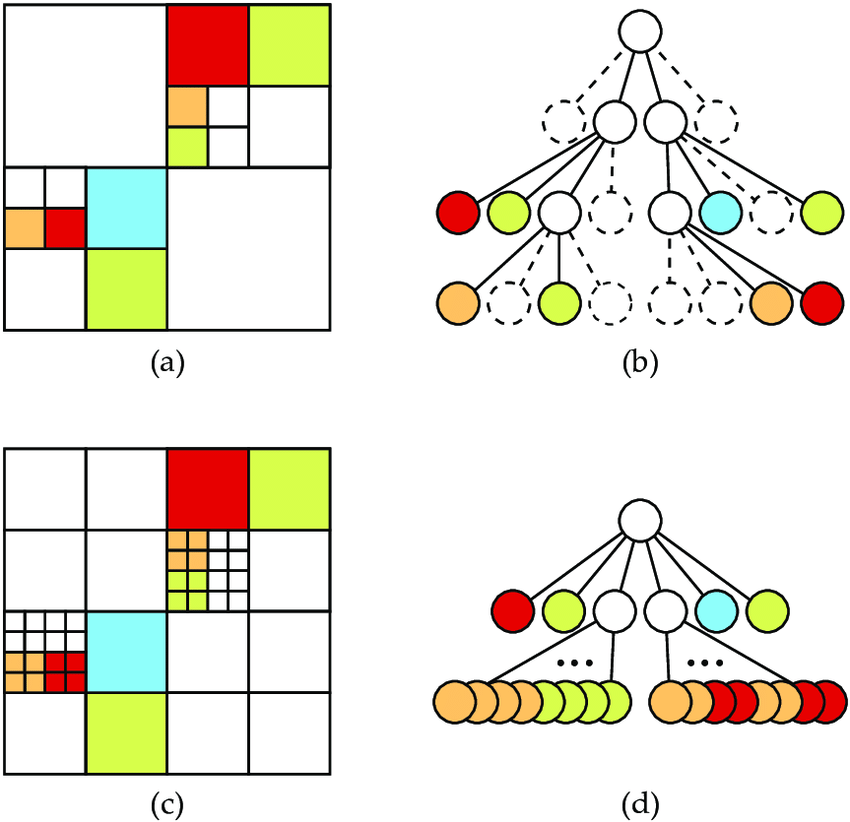


Рисунок .. Примеры построения октодерева []

Для создания воксельного объекта используются вокселезация -преобразование объекта, состоящего из полигонов, в набор вокселей []. 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 [].

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:
   1. Определяется позиция и масштаб деформации.
   2. Пока соответствующий лист SVO больше масштаба деформации – детализируем данную ветвь SVO.
   3. 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 (табл. 1.2):

Таблица .. Сравнительная таблица методов деформации ландшафта

| **Критерий** | **Полигональная сетка** | **Воксели** |
| --- | --- | --- |
| Влияние на геймплей | Внутренность объекта не важна | Составляют полный объём объекта |
| Аппаратная часть | GPU | CPU |
| Как деформирует ландшафт? | Сетка детализируется в месте деформации, для каждого треугольника рассчитывается новая позиция | Добавление и удаление вокселей в месте деформации |
| Основная проблема | Большое количество расчетов при деформации | Нагрузка на память при большом разрешении игрового мира |
| Насколько эффективны методы оптимизации | вместо со структурами данных (где – требуемое количество полигонов для отрисовки) [6] | Зависит от вокселезируемого объекта, в приведенных выше примерах затраты памяти ниже в 3.5 раза и 5.5 раз с использованием SVO [12] |

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

На основе рассмотренных ранее игровых аналогов можно выделить следующие общие паттерны жанра шутер:

1. Игрок использует оружие дальнего боя, чтобы наносить урон противникам и деформировать ландшафт.
2. Игрок может использовать разное оружие дальнего боя, может менять его во время игры.
3. Используется камера от первого лица.

В нашей версии будут добавлены следующие механики:

1. Игра происходит в реальном времени.
2. В игре может участвовать несколько игроков одновременно.
3. Все игроки могут деформировать ландшафт.
4. Разные оружия наносят разный урон противникам и по-разному деформируют ландшафт (изменяется вид кратера от взрыва, его размеры и тому подобное).
5. Игроки могут деформировать любое игровое окружение, включая какие-либо статические объекты на карте.
6. Использование воксельного метода генерации ландшафта позволит для каждого вокселя задавать дополнительную информацию. На основе этого, возможна генерация объектов и локаций из различного материала, и изменение уровня их деформируемости.

### 1.3.1. Описание функциональных и нефункциональных требований к игре

**Назначение приложения**: решение проблемы оптимизации отрисовки изображения при деформации ландшафта в игре жанра шутер.

На основе проанализированных существующих игровых решений и методов генерации деформируемых ландшафтов, выявим следующие требования.

**Функциональные требования**:

1. Функциональные требования к игре:
   1. Возможность изменить настройки игры.
      1. Возможность изменить уровень звука в игре.
      2. Возможность изменить уровень разрешения экрана в игре.
      3. Возможность изменить качество текстур в игровой миссии.
      4. Возможность изменить в настройках управление персонажем.
2. Функциональные требования к игровому процессу:
   1. Персонаж игрока может перемещаться по игровой карте.
   2. Персонаж игрока может производить стрельбу из своего оружия.
   3. При получении определенного количества урона персонаж игрока погибает и теряет способность к передвижению и ведению стрельбы.
   4. Игрок может производить перезарядку своего оружия.
   5. Игрок может менять оружие, из которого производится стрельба

**Нефункциональные требования**:

1. Игровая камера находится в режиме от первого лица.
2. Минимальное количество кадров в секунду – 30.

**Требования к пользовательскому интерфейсу**:

1. Пользовательский интерфейс представлен на английском языке.

# 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

Выделив основные паттерны и основные игровые механики жанра «шутер», приведём их декомпозицию с целью выявления связей между компонентами и схемы взаимодействия подсистем. Для этого составим диаграммы прецедентов и опишем сценарии взаимодействия компонентов (см. рис. 2.1).

1. Название: Изменение пользователем игровых настроек (табл. 2.1).

Акторы: игрок.

Краткое описание: игрок хочет изменить игровые настройки, чтобы игровая производительность соответствовала возможностям компьютера.

Триггер: запуск приложения.

Таблица 2.1. Описание прецедента "Изменение пользователем игровых настроек"

| Действия акторов | Отклик системы |
| --- | --- |
| 1) Игрок запускает систему. | 2) Система запускается и открывает главное меню. |
| 3) Игрок выбирает пункт меню «Настройки». | 4) Система открывает окно «Настройки» |
| 5) Игрок меняет игровые настройки и нажимает кнопку «Принять». | 5) Система обрабатывает введённые данные, изменяя настройки в соответствии с введёнными (Е1). |

Альтернативные потоки:

E1: в случае если настройки не были изменены, их перенастройка не производится.

2. Название: Перемещение персонажа по игровой карте (см. табл. 2.2).

Акторы: игрок.

Краткое описание: игрок хочет изменить своё местоположение на игровой карте.

Триггер: запуск игровой миссии.

Таблица 2.2. Описание прецедента "Перемещение персонажа по игровой карте"

| Действия акторов | Отклик системы |
| --- | --- |
| 1) Игрок запускает игровую миссию. | 2) Система запускает игровую миссию, генерирует ландшафт и создаёт на ландшафте персонажей. |
| 3) Игрок нажимает клавиши, отвечающий за передвижение персонажа. | 4) Система расчитывает координаты, в которые должен переместиться персонаж, и персонаж начинает перемещаться туда по проложенной траектории (Е1). |

Альтернативные потоки:

E1: в случае если рассчитанная координата недосягаема для персонажа, персонаж проходит по траектории до конечной досягаемой координаты и останавливается.

3. Название: Смена оружия (табл. 2.3).

Акторы: игрок.

Краткое описание: игрок может сменить оружие, чтобы изменить тип снарядов.

Триггер: запуск игровой миссии.

Таблица 2.3. Описание прецедента "Смена оружия"

| Действия акторов | Отклик системы |
| --- | --- |
| 1) Игрок нажимает клавишу для смены оружия. | 2) Система меняет оружие, принадлежащее персонажу в данный момент. |

4. Название: Ведение стрельбы персонажем (табл. 2.4).

Акторы: игрок.

Краткое описание: игрок хочет начать вести стрельбу в игровой миссии, по ландшафту или по противнику.

Триггер: прецедент «Смена оружия».

Таблица 2.4. Описание прецедента "Ведение стрельбы персонажем"

| Действия акторов | Отклик системы |
| --- | --- |
| 1) Игрок запускает игровую миссию. | 2) Система запускает игровую миссию, генерирует ландшафт и создаёт на ландшафте персонажей. |
| 3) Игрок наводит камеру в нужное ему место и нажимает клавишу, отвечающую за стрельбу. | 4) Система расчитывает координаты, куда должен полететь снаряд и запускает его по траектории. Если при достижении одной из точек траектории там находился персонаж противника, ему наносится урон в соответствии с типом снаряда (Е1). |

Альтернативные потоки:

E1: в случае если при достижении одной из точек траектории там находился ландшафт, он деформируется в соответствии с типом снаряда.

5. Название: Смерть игрового персонажа (табл. 2.5).

Акторы: игрок.

Краткое описание: когда персонаж получает урона больше, чем он имеет здоровья, он погибает и теряет способность к ведению любой деятельности.

Триггер: прецедент «Ведение стрельбы персонажем».

Таблица 2.5. Описание прецедента "Смерть игрового персонажа"

| Действия акторов | Отклик системы |
| --- | --- |
| 1) Игрок наводит камеру под таким углом, чтобы попасть в противника и нажимает клавишу стрельбы. | 2) Система расчитывает координаты, куда должен полететь снаряд и запускает его по траектории. Снаряд попадает в персонажа противника и наносит ему урон. Если у противника заканчиваются очки здоровья, система перестаёт реагировать на его действия и считает его «мёртвым». |

6. Название: Перезарядка оружия (табл. 2.6).

Акторы: игрок.

Краткое описание: когда у игрока кончаются снаряды, доступные для стрельбы, он хочет провести перезарядку для увеличения количества снарядов для использования.

Триггер: прецедент «Ведение стрельбы персонажем».

Таблица 2.6. Описание прецедента "Перезарядка оружия"

| Действия акторов | Отклик системы |
| --- | --- |
| 1) Игрок нажимает клавишу для перезарядки. | 2) Система обновляет количество снарядов, доступных для стрельбы. |

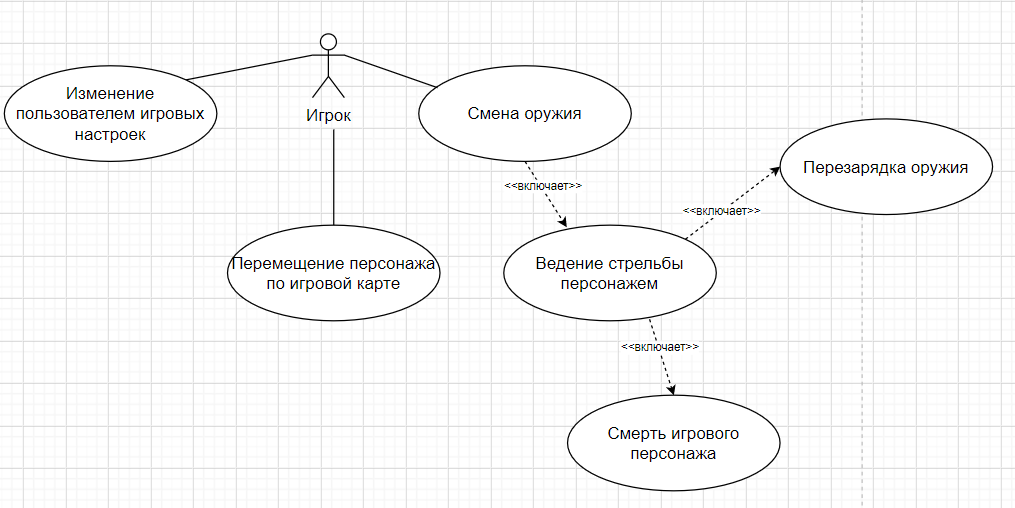


Рисунок .. Диаграмма прецедентов

На этой основе составим диаграммы последовательностей для более детального понимания взаимодействия между подсистемами при действиях пользователя.

1) Имя: «Изменить игровые настройки» (рис. 2.2)

Обязанности: нет.

Ссылки: прецедент «Изменение пользователем игровых настроек».

Исключения: если пользователь несколько раз поменял значения, и в итоге они остались старыми, перенастройка не проводится.

Предусловия: включение приложения, переход в раздел главного меню «Настройки».

Постусловия: изменение значений настроек. Игра будет использовать новые настройки.

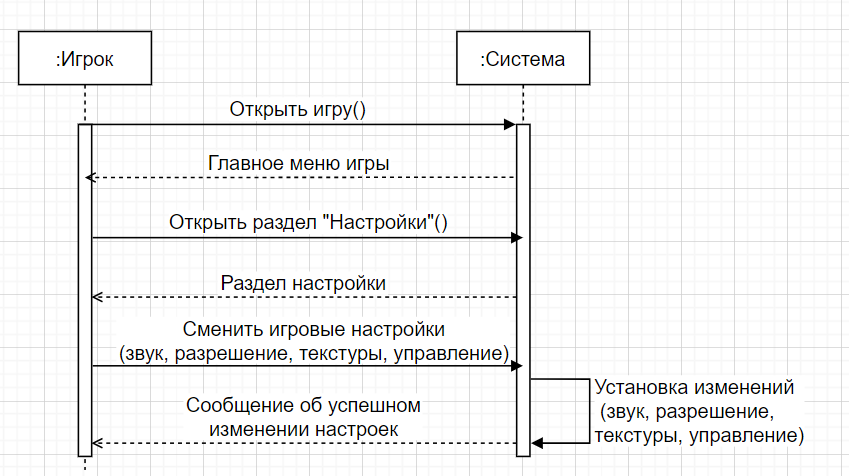


Рисунок .. Диаграмма последовательностей "Изменить игровые настройки"

2) Имя: «Переместить персонажа на игровой карте» (рис. 2.3)

Обязанности: персонаж должен быть «живым», то есть уровень здоровья персонажа выше нуля.

Ссылки: прецедент «Перемещение персонажа по игровой карте».

Исключения: если пользователь старается переместиться в недоступную для него координату, то он останавливается в ближайшей доступной точки на проложенной траектории.

Предусловия: включение приложения, включение игры.

Постусловия: изменённое положение персонажа на игровой карте.

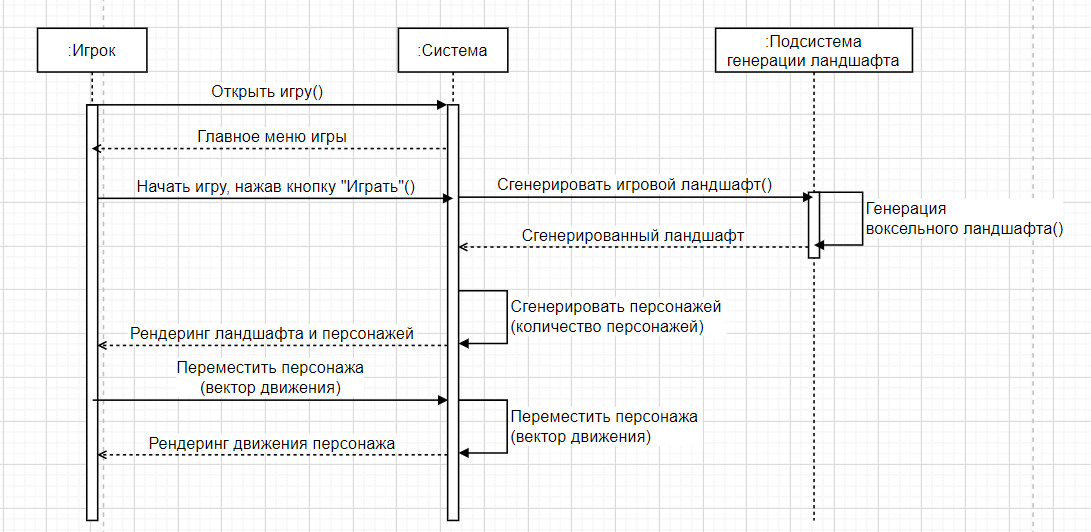


Рисунок .. Описание последовательностей "Переместить персонажа на игровой карте"

3) Имя: «Сменить оружие» (рис. 2.4)

Обязанности: персонаж должен быть «живым», то есть уровень здоровья персонажа выше нуля.

Ссылки: прецедент «Смена оружия».

Исключения: нет.

Предусловия: включение приложения, включение игры.

Постусловия: изменённый тип оружия у игрока.



Рисунок .. Описание последовательностей "Сменить оружие"

4) Имя: «Провести стрельбу» (рис. 2.5)

Обязанности: персонаж должен быть «живым», то есть уровень здоровья персонажа выше нуля.

Ссылки: прецедент «Ведение стрельбы персонажем».

Исключения: если у игрока нет снарядов, доступных для стрельбы, стрельба не ведётся.

Предусловия: включение приложения, включение игры.

Постусловия: изменённое количество снарядов у игрока, изменённое игровое состояние в зависимости от попадания снаряда.

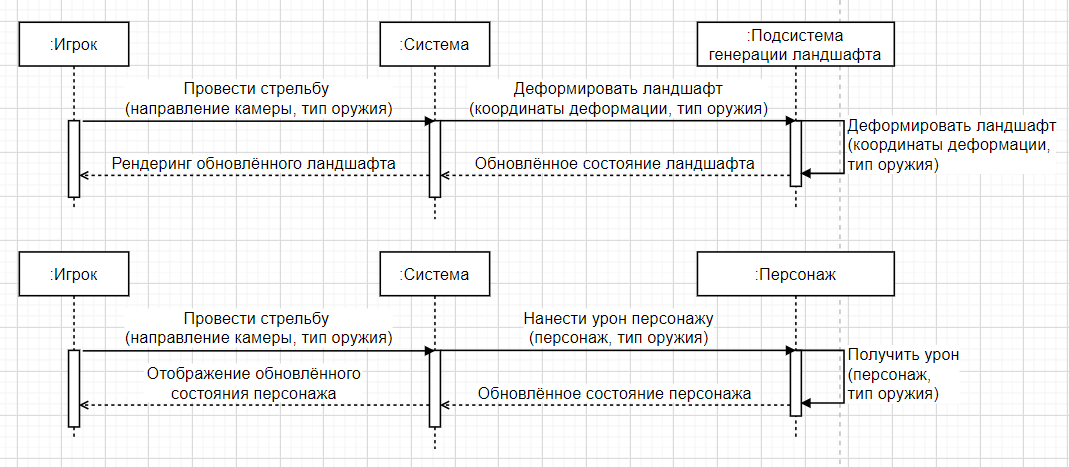


Рисунок .. Описание последовательностей "Провести стрельбу"

5) Имя: «Отключить персонажа по причине смерти» (см. рис. 2.6)

Обязанности: уровень здоровья персонажа должен опуститься ниже нуля.

Ссылки: прецедент «Смерть персонажа».

Исключения: нет.

Предусловия: попадание одного или более снарядов противников в персонажа.

Постусловия: изменённое состояние игры, изменённое состояние персонажа.

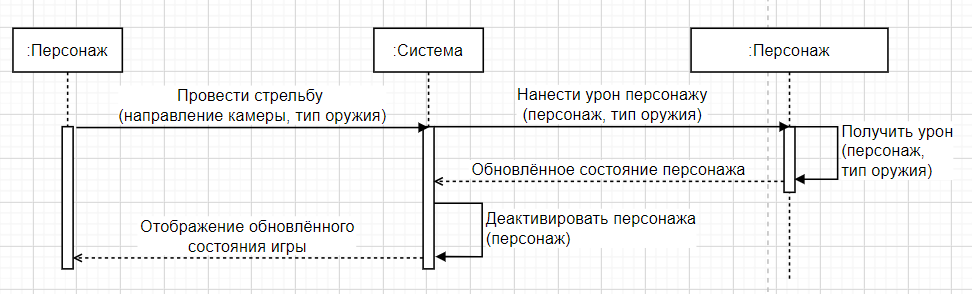


Рисунок .. Описание последовательностей "Отключить персонажа по причине смерти"

6) Имя: «Перезарядить оружие» (рис. 2.7)

Обязанности: количество доступных для стрельбы снарядов равно нулю.

Ссылки: прецедент «Перезарядка оружия».

Исключения: нет.

Предусловия: проведение серии выстрелов персонажем.

Постусловия: изменённое количество снарядов, доступных для стрельбы.

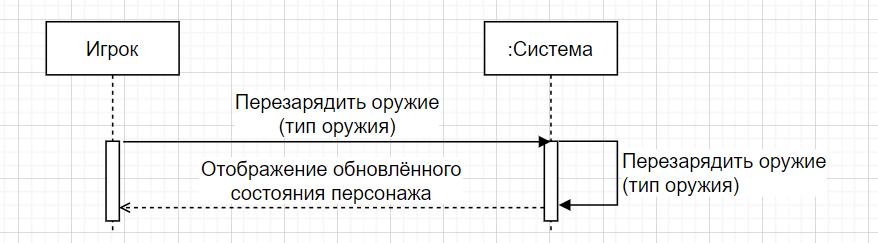


Рисунок .. Описание последовательностей "Перезарядить оружие"

Как мы видим, дополнительно требуется описать некоторые подсистемы. Требуется проектирование алгоритма работы подсистемы генерации деформируемого ландшафта и проектирование жизненного цикла персонажа. Затем, требуется описать взаимодействие этих двух подсистем через взрывы снарядов.

## 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.

# 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?

# 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. Игры с лучшей разрушаемостью [Электронный ресурс]. – Режим доступа: <https://cubiq.ru/igry-s-luchshej-razrushaemostyu/> (дата обращения 01.03.2021).
3. Scorched 3D Version 43.2 beta released [Электронный ресурс]. – Режим доступа: <https://www.webcitation.org/666BwPZfZ?url=http://scorched3d.co.uk/phpBB3/viewtopic.php?t=6268> (дата обращения 13.02.2021).
4. Terrain in Battlefield 3: A modern, complete and scalable system [Электронный ресурс]. – Режим доступа: <https://media.contentapi.ea.com/content/dam/eacom/frostbite/files/gdc12-terrain-in-battlefield3.pdf> (дата обращения 13.02.2021).
5. Battlefield 3. Планы на переворот [Электронный ресурс]. – Режим доступа: <http://mgnews.ru/read-news/battlefield-3-plany-na-perevorot> (дата обращения 13.02.2021).
6. Gregory J. Game Engine Architecture – 3rd ed. – Boca Raton.:CRC Press, 2018. – P. 525-544.
7. Mistal, B. GPU Terrain Subdivision and Tessellation // GPU Pro. – 2018. – Vol. 360. – P. 177-194.
8. Pangerl, D. Dynamic GPU Terrain // GPU Pro. – 2018. – Vol. 360. – P. 223-237.
9. Akenine-Moller, T., Haines, E. Real-time Rendering – 4th ed. – Boca Raton.:CRC Press, 2018. – P.545-589.
10. 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).
11. NVIDIA Blast [Электронный ресурс]. – Режим доступа: <https://developer.nvidia.com/blast> (дата обращения 22.01.2021).
12. Pätzold, M. Graphical model and simulation for THz-imaging: dis. Dok. der Ing. – Siegein, 2017. – S. 9.
13. Schawaz, M., Seidel, H. P. Fast Parallel Surface and Solid Voxelization on GPUs // ACM Transations on Graphics. – 2010. – Vol.29. – P.179:1-179:10.
14. Baert, J., Lagae, A., Dutre, Ph. Out-of-Core Construction of Sparse Voxel Octrees // Computer Graphics Forum. – 2013. – Vol. 33. – P.220-227.
15. Kotfis, D., Cozzi, P. Octree Mapping from a Depth Camera // GPU Pro. – 2016. – Vol. 7. – P. 257-273.
16. Palmer, G. Physics for Game Programmers – New York.:Apress, 2018. – P. 83-138.