Санкт-Петербургский государственный университет

Математико-механический факультет
Математическое обеспечение и администрирование информационных систем

Гурьев Василий Александрович

Оптимизация построения и хранения lightmap

Курсовая работа

Научный руководитель:
Пименов А. А.

Санкт-Петербург
2019

Оглавление

Введение

Данная работа посвящена исследованию и реализации различных алгоритмов оптимизации построения и хранения lightmap (карты освещения). Lightmap - метод моделирования освещенности объектов в 3D приложениях. Он заключается в точном просчете освещенности каждого статичного объекта внутри сцены до запуска приложения и сохранении этих данных в текстуру.

Базовая версия алгоритма работает следующим образом - создается текстура, каждый текстель которой хранит информацию об освещенности определенной точки на полигоне. Изначально текстура заполняется черным цветом. Затем для каждого полигона считается освещение от статичных (остающихся на месте) источников света. При этом учитывается, что точка может быть в тени, для этого строится вектор от точки до каждого из источников света, проверяется пересечение с геометрией сцены. Так же обычно учитывается затухание света, всевозможные отражения, преломления и рассеивания. После этого значение текстеля карты освещенности увеличивается на посчитанную величину. Эта процедура повторяется для всех полигонов сцены. Для просчета света можно использовать различные модели освещения (например модель освещения Фонга), для данного алгоритма это не имеет значения. Далее после записи всех значений в lightmap каждому полигону сцены присваиваются 2 значения - так называемые uv координаты. Это координаты точки на карте освещенности. И потом, в процессе отрисовки всех полигонов, значения из lightmap применяются для получения финальной картины освещения сцены.

На данный момент этот алгоритм применяется в большинстве 3D приложений, так как позволяет значительно увеличить производительность приложения. Так же из-за того, что свет просчитывается заранее, считается он достаточно качественно. Это позволяет получить качество картинки намного выше, чем если бы освещенность считалась перед каждым кадром, как это делается для динамических объектов и для динамических источников освещения.

Целью данной работы является попытка оптимизации построения и хранения lightmap в крупном игровом проекте [3]. Для успешного выполнения необходимо:

- Исследовать текущую реализацию lightmap.
- Исследовать реализацию lightmap в других проектах.
- Пользуясь данными исследованиями придумать возможную оптимизацию текущего решения.
- Внедрить данную оптимизацию в проект.

1. Существующее решение

До того, как началась данная работа, в проекте World War Z была реализована одна из модификаций lightmap. Вместо сохранения 1 текстуры карты освещения сохраняется 10 на каждый уровень. 5 текстур сохраняется для высокого качества настройки графики, 5 для низкого (в проекте на разных настройках графики разные конфигурации источников освещения, а карты освещенности соответственно различаются). Из этих 5:

- Текстура, для каждого полигона содержащая направление на наиболее сильный источник освещения. Обычно это направление на солнце. Данная информация нужна для реализации всевозможных графических эффектов.

- Текстура, для каждого полигона содержащая освещенность от этого наиболее сильного источника.

- 3 текстуры, содержащие информацию о том, как была бы освещена данная точка, если бы ее нормаль была повернута относительно ее реальной нормали. То есть в данной точке строятся 3 вектора, образующие правильную треугольную пирамиду, где реальная нормаль - высота пирамиды. Данная информация нужна, чтобы при любых искажениях нормали можно было разложить текущую нормаль по 3 данным векторам и вычислить корректное значение освещенности. А искажения нормали происходят опять же при различных графических эффектах.

То есть для каждого полигона хранятся одни uv - координаты, всего 2 значения, по ним можно из карт освещенности получить доступ к 30 значениям типа float - 10 текстур, в каждой точке которых хранятся 3 значения (3 канала цвета - red, green, blue).

При этом для достаточно большой сцены каждая из этих текстур получается размером примерно 3500x3500 точек, несжатая версия таких текстур весит около 50 мегабайт, 10 текстур - половина гигабайта,

что слишком много для всего одного уровня. Поэтому после просчета lightmap-а текстуры сжимаются в формат BC6 или BC7, зависит от текстуры. Данные форматы реализованы в DirectX 11 с поддержкой быстрого чтения из сжатых текстур, более подробно с ними можно ознакомиться тут [1]. Главное - данные форматы обладают постоянной степенью сжатия, однако искажение, получаемое при сжатии, меньше всего, если в каждом блоке 4x4 текстелей хранятся похожие значения. То есть каждая текстура делится подряд на квадраты 4x4, чем ближе значения внутри каждого блока, тем ближе сжатое изображение к несжатому.

Из-за таких особенностей сжатия после построения всех текстур делается перестановка точек внутри них таким образом, чтобы похожие были внутри квадратов 4x4. Но так как uv координаты для каждого полигона одни на все 10 текстур, похожие точки ищутся не в 3-мерном, а в 30-мерном пространстве.

В конечном виде текстуры приобретают следующий вид:

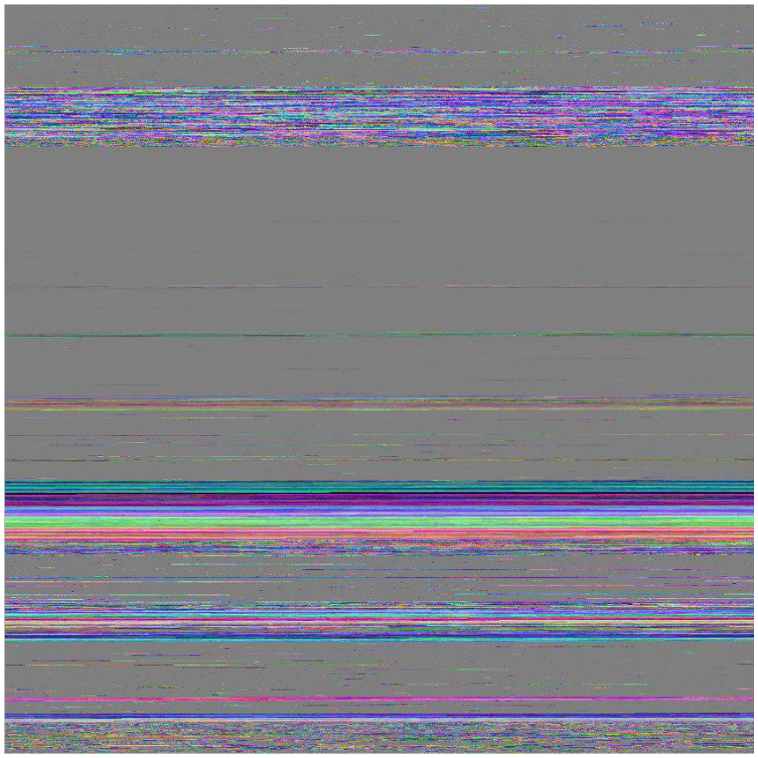

Рис. 1: Текстура направлений на сильнейший источник света

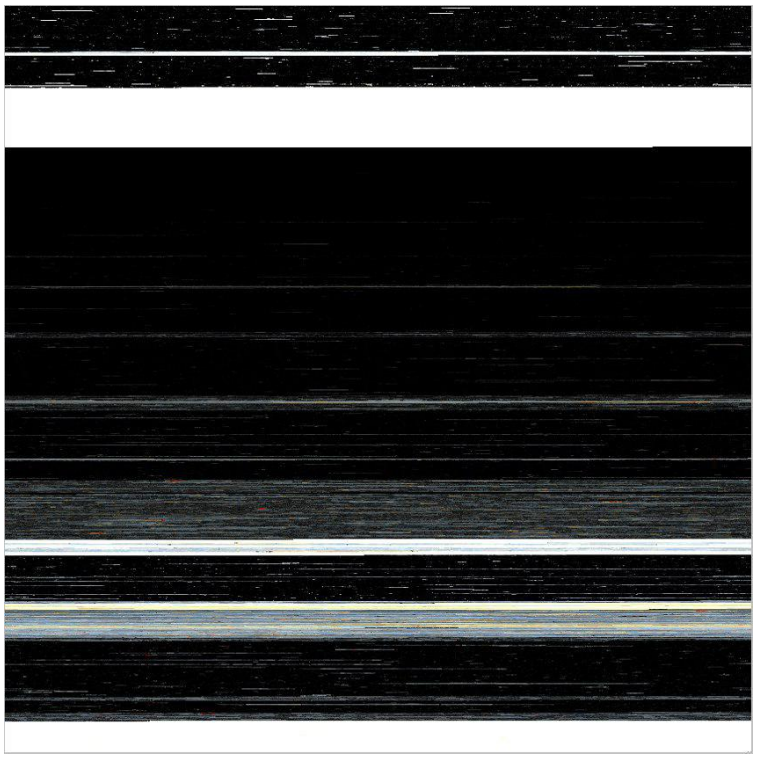

Рис. 2: Текстура цвета сильнейшего источника света

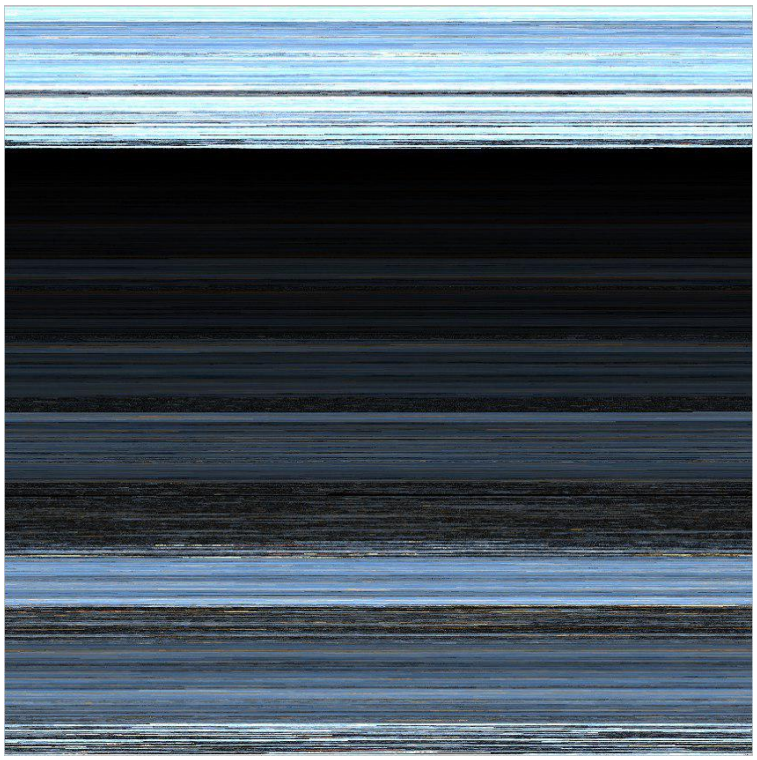

Рис. 3: Текстуры освещения относительно повернутых нормалей, все 3 имеют примерно одинаковый вид

2. Оптимизация

Первый шаг в оптимизации данного алгоритма - попытаться еще сильнее сжать конечные текстуры.

Для поиска подходов к этому была написана программа, принимающая на вход 10 текстур и собирающая о них всевозможную статистику. Прежде всего она измеряла, насколько похожие значения находятся внутри каждой ячейки 4x4. Для этого она строила 4 графика. На каждом из них по оси X отложено то, за что отвечает данный график, по оси Y - количество ячеек в lightmap. Например из графика среднеквадратичного отклонения можно понять, что среднеквадратичное отклонение от 0 до 0.017 имеют чуть меньше 130 тысяч ячеек, от 0.017 до 0.034 около 46 тысяч ячеек и т.д.

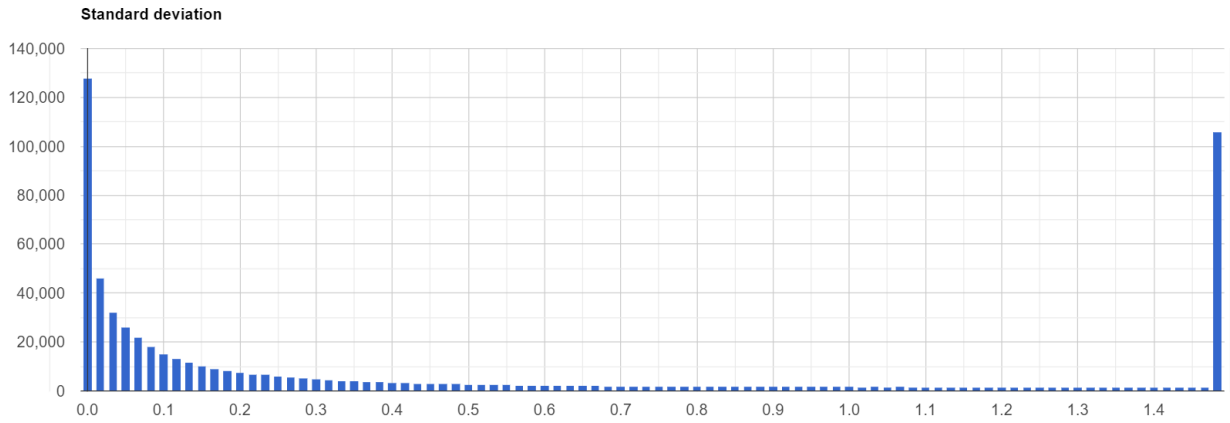

Рис. 4: Среднеквадратичное отклонение точек от среднего значения внутри ячейки 4x4

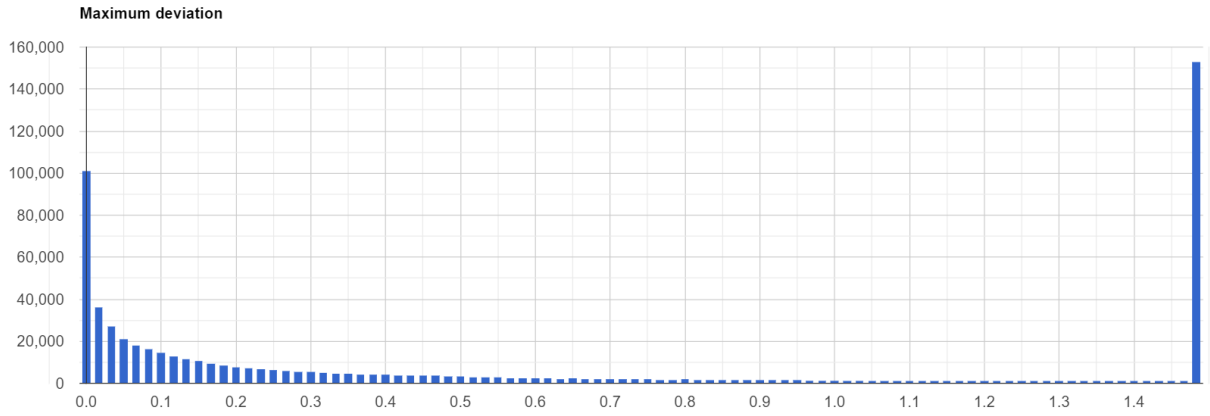

Рис. 5: Максимальное квадратичное отклонение точек от среднего значения внутри ячейки 4x4

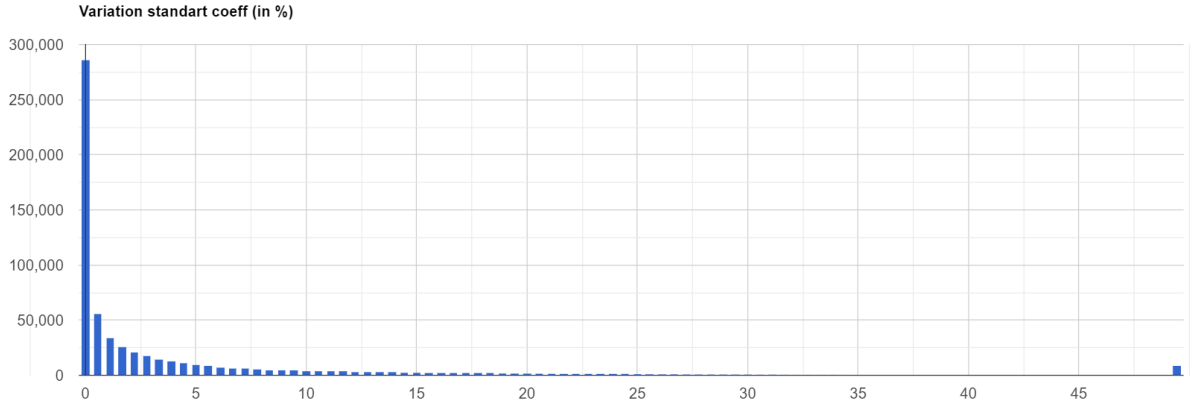

Рис. 6: Среднее относительное отклонение точек от среднего значения в процентах внутри ячейки 4x4

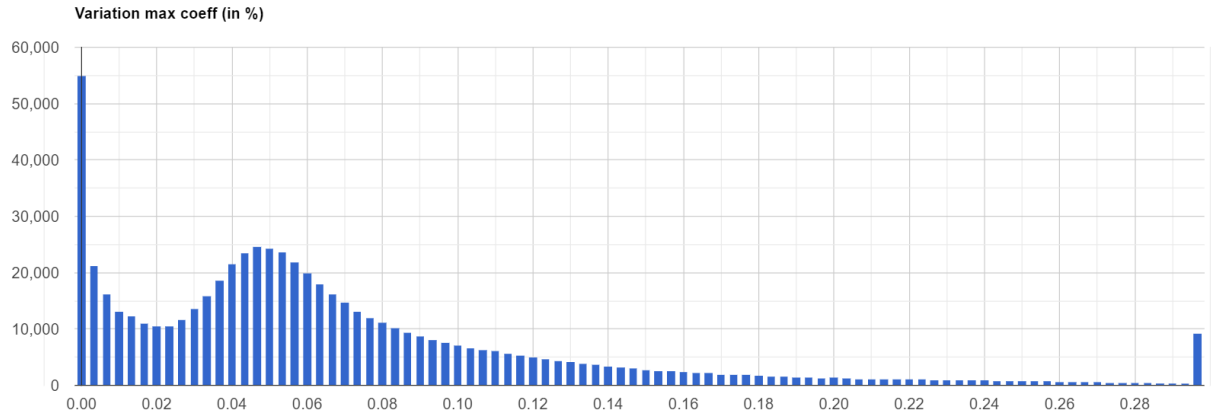

Рис. 7: Максимальное относительное отклонение точек от среднего значения внутри ячейки 4x4, приведенное к виду от 0 до 1 при помощи преобразования x / (x + 1)

После построения данных графиков стало понятно, что значения внутри ячеек достаточно похожи и некоторые точки lightmap-а можно объединить в одну, практически не повлияв на качество итоговой карты освещенности, зато достаточно сильно сократив объем занимаемой им памяти. То есть у точек на полигонах с похожим освещением можно поставить одинаковые uv координаты. Чтобы понять, какие точки можно объединять и по какому принципу это лучше всего делать, была проведена серия экспериментов. По каждому из графиков брались несколько значений на оси X и все ячейки 4x4, лежащие слева от этих значений, целиком перекрашивались в среднее значение по этой ячейке, затем загружался уровень и искались изменения. То есть имитировался подход с объединением некоторых точек в одну для нахождения критерия, какие точки можно объединять без сильной потери качества итоговой карты освещения.

Подходы со среднеквадратичным отклонением и средним относительным отклонением были забракованы, так как при их использовании при объединении N точек если среди них N-1 практически одинаковые, а одна совершенно другая, то среднее отклонение получается достаточно маленькое и эта одна точка сильно преобразуется относительно

настоящей карты освещения, эпизодически давая визуальные ошибки освещения. Подход с максимальным квадратичным отклонением так же работал достаточно плохо, так как оно одинаковое для освещения в тени и на свету. Пример - если значения освещенности около 10, то квадратичное отклонение в 0.01 невозможно заметить, но если значение освещенности 0.05, то изменение на 0.01 сильно бросается в глаза и дает визуальные баги. Критерий максимального относительного отклонения всех точек от среднего этих точек меньше некоторого заранее выбранного EPS работал достаточно хорошо, но его все равно стоило доработать для наилучшего результата.

Итого критерий объединения 2 точек по результатам множества экспериментов было решено выбрать такой - если по каждой из 30 координат эти значения отличаются не более чем на 1% от среднего получившегося значения, то данные точки могут быть объединены в одну, к которой в свою очередь данный критерий уже неприменим. Данный критерий позволял заменять достаточно много ячеек одной точкой, при этом не получая никаких визуальных багов.

После нахождения оптимального критерия было решено перед разбиением точек на похожие ячейки, для последующего сжатия в один из форматов BC6/BC7, объединить все возможные точки по данному критерию. Но к этому алгоритму было достаточно жесткое ограничение - он должен работать не больше 10 минут на данных в 12 миллионов 30-ти мерных точек. По этой причине наивный алгоритм с квадратичной сложностью был отброшен, как непригодный.

Был разработан следующий достаточно точный алгоритм - по каждой из 30 координат строился массив пар (значение; индекс точки), массивы сортировались по значениям. Далее берется случайный индекс точки, на основании критерия по каждому из массивов бинарным поиском ищется Eps-окрестность этой точки и получается 30 списков индексов. Их пересечение и есть искомое множество, которое можно объединить. Данный алгоритм работает за O(N * (M + log(N))), где N - количество точек, M - среднее количество точек внутри Eps-окрестности по всем координатам. Но на практике оказалось, что это все равно до-

статочно долго, на 12 миллионах точек алгоритм работал примерно 5 часов. Зато по оптимизации памяти он показал неплохой результат - уменьшил количество точек на 40.6%.

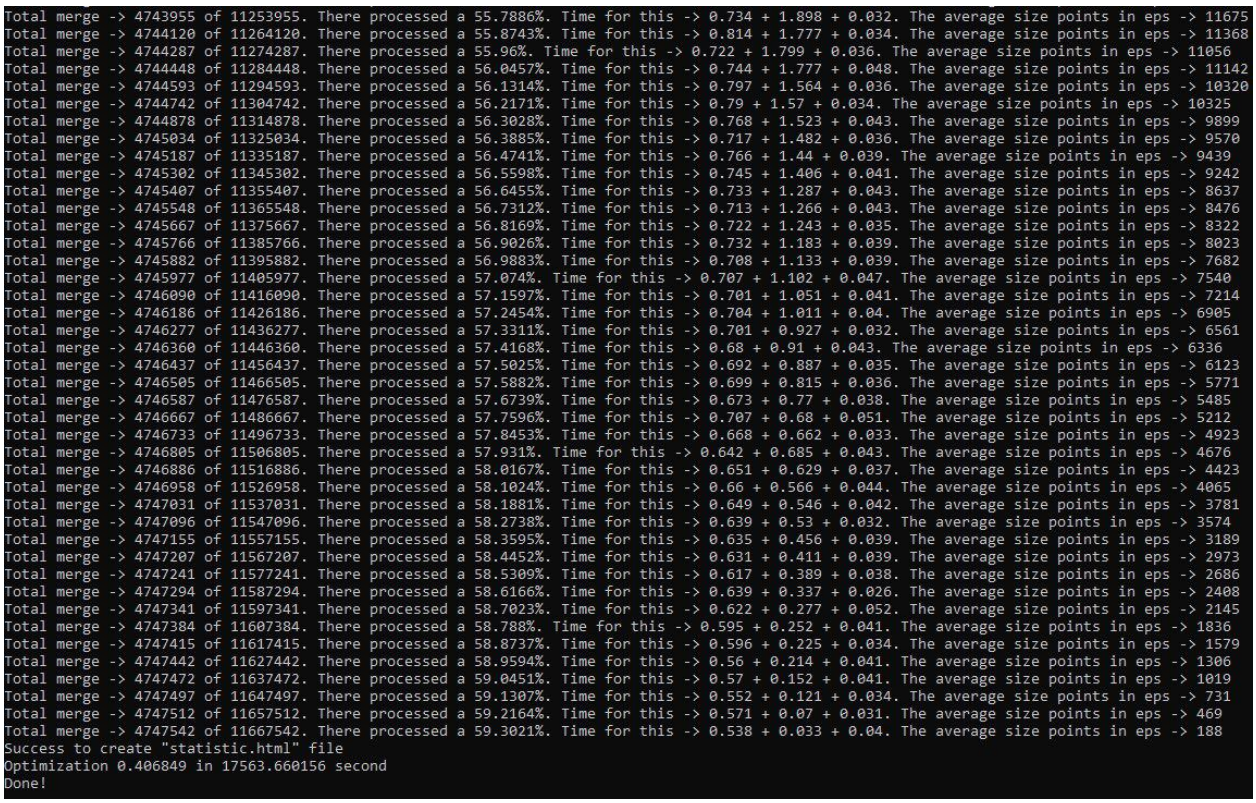

Рис. 8: Статистика работы алгоритма

Большая проблема данного алгоритма в том, что в Eps-окрестность по каждой координате попадает слишком много точек, что сильно замедляет время работы. Было найдено следующее решение - разбить входные точки на схожие кластеры не слишком большого размера и параллельно запустить данный алгоритм на найденных множествах. В качестве алгоритма кластеризации был выбран достаточно быстрый и популярный алгоритм kmeans, вернее его немного улучшенная версия kmeans++ [2]. Данный алгоритм был выбран из-за его быстрой сходимости к неплохого качества кластерам, он нужен только для быстрого начального разбиения. То есть kmeans++ разбивал входные данные на кластеры размером не больше 100 тысяч точек, после чего на данных кластерах параллельно запускался выше описанный алгоритм. Это да-

ло хорошие результаты - уменьшение количества точек на 38.5% за 8 минут, при этом из этих 8 минут около 5 занимает кластеризация и около 3 - параллельная обработка кластеров алгоритмом.

3. Оптимизация 2

В процессе исследования алгоритма были разработаны и испытаны еще несколько идей оптимизации. В их числе - хранение разных uv-координат для разного качества игры, высокого и низкого. Так как в низком качестве присутствуют дополнительные источники света, сохраняемые в lightmap (в высоком уровне качества они оставлены динамическими и освещение от них считается каждый кадр), то карт освещения на уровень получается 10, а не 5. Но если хранить для каждого полигона на геометрии дополнительные uv-координаты и переключать их при смене качества, то можно сказать, что использоваться будет 5 текстур и предыдущая оптимизация будет искать похожие точки не в 30-ти мерном пространстве, а в 15-ти. Была проведена серия замеров, определяющая, на сколько при этом сократится размер lightmap и сколько будет выиграно по памяти. По результатам замеров от данной оптимизации было решено отказаться, так как она требует хранения дополнительных данных для каждого полигона, а переход из 30-мерного пространства в 15-мерное дал не слишком большой прирост по сохраненной памяти.

4. Результаты оптимизации

Итак, был написан алгоритм, сокращающий расход памяти на 38.5% всего за 8 минут предварительных вычислений. Язык написание всего кода - C++. После этого была проведена все та же серия тестов - запуск уровня с разными картами освещенности и сравнение изображений. Следующие скриншоты сделаны в режиме "only lighting", то есть не учитывается ничего, кроме цвета освещенности точки. Первый скриншот - красным помечено, какие точки внутри карты освещенности поменялись, второй - то, что было до оптимизации, третий - то, что стало после. Разницу между 2 и 3 скриншотами невооруженным глазом заметить очень сложно, что является показателем качества оптимизации. (Лучи за шкафом на последней серии скриншотов делаются динамическим освещением и никак не связаны с картой освещенности)

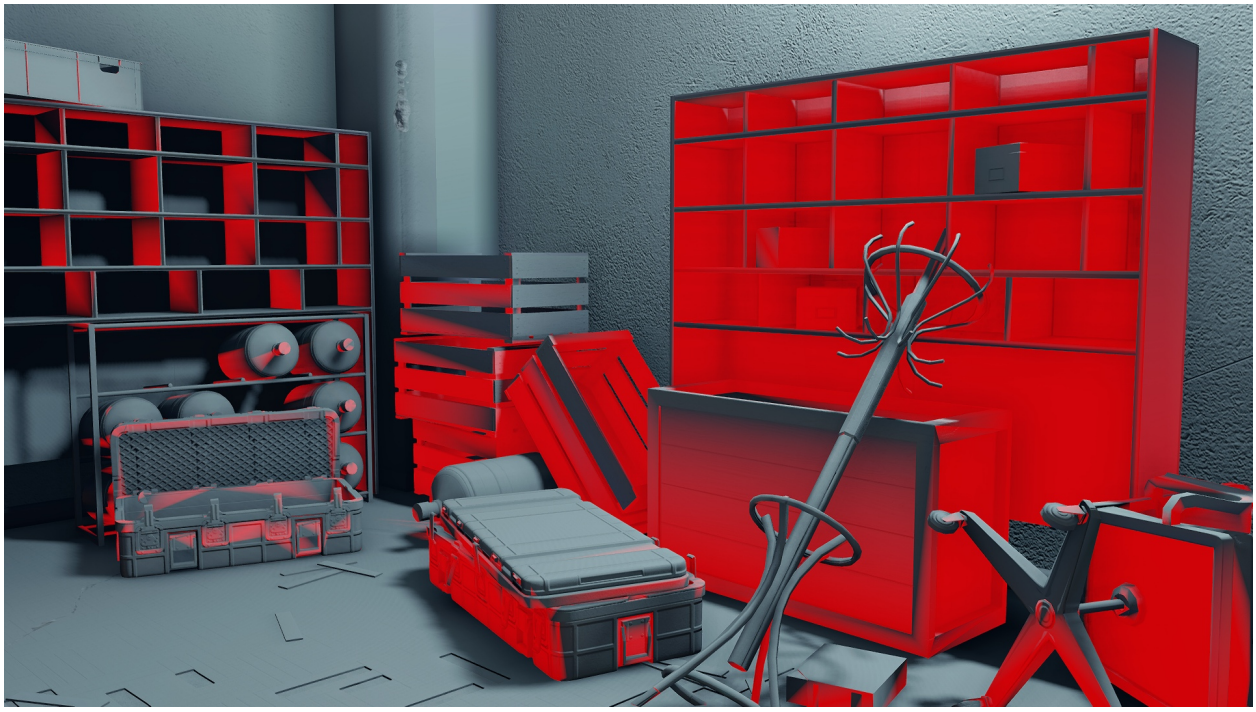

Рис. 9: Изменения

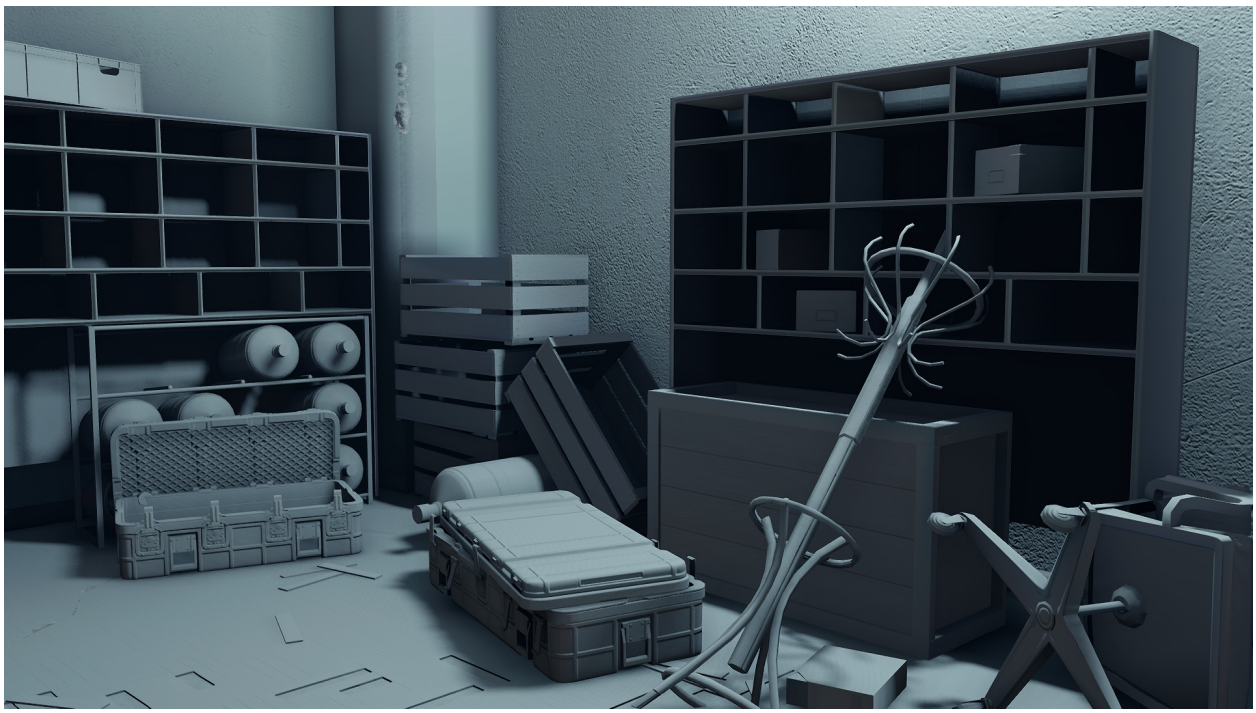

Рис. 10: До работы алгоритма

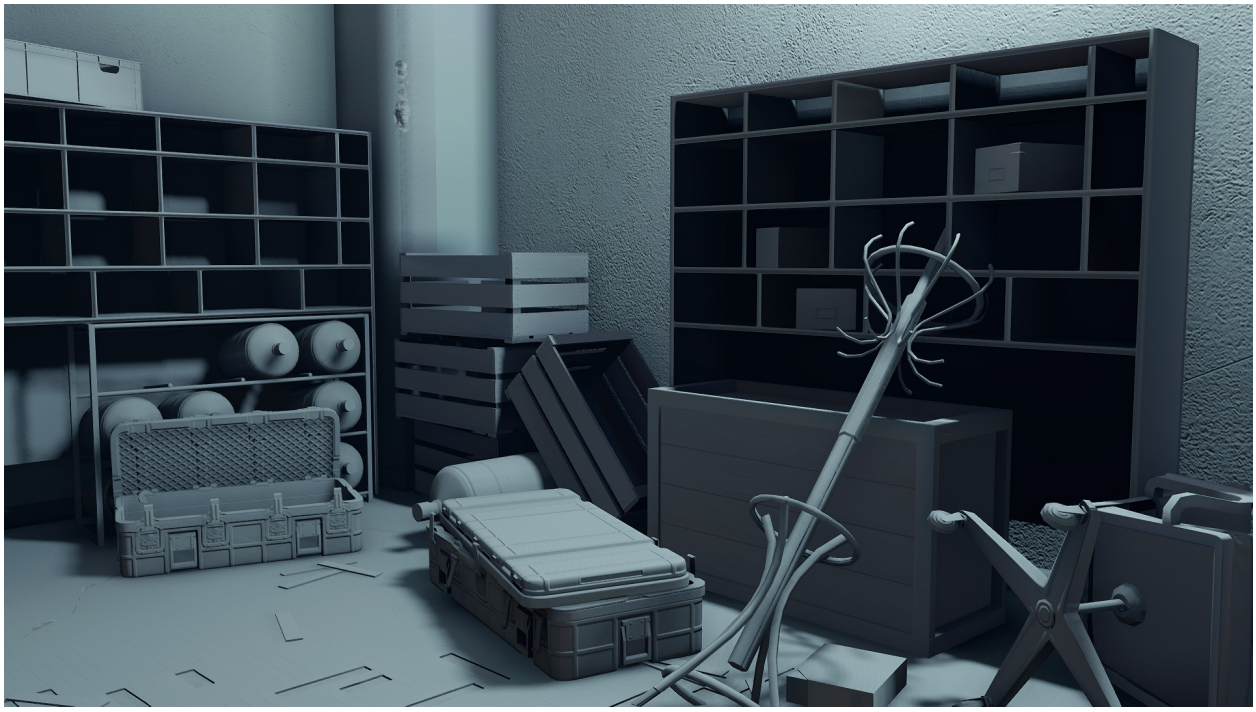

Рис. 11: После работы алгоритма

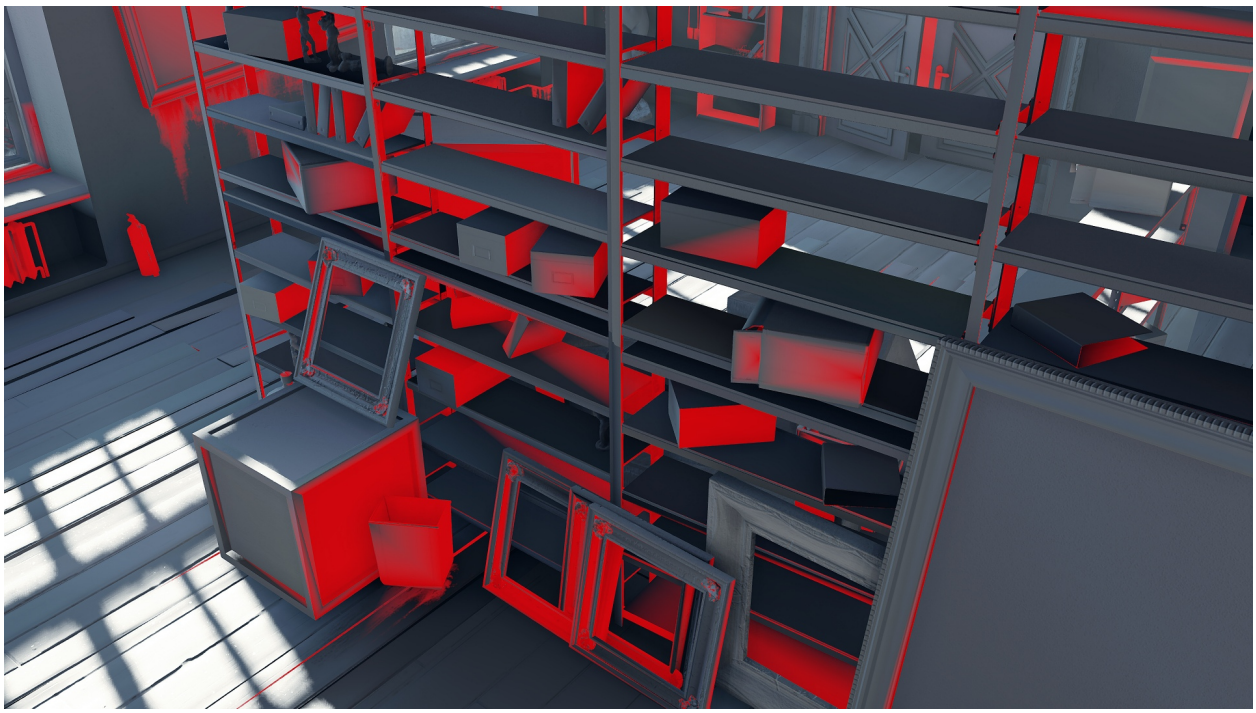

Рис. 12: Было

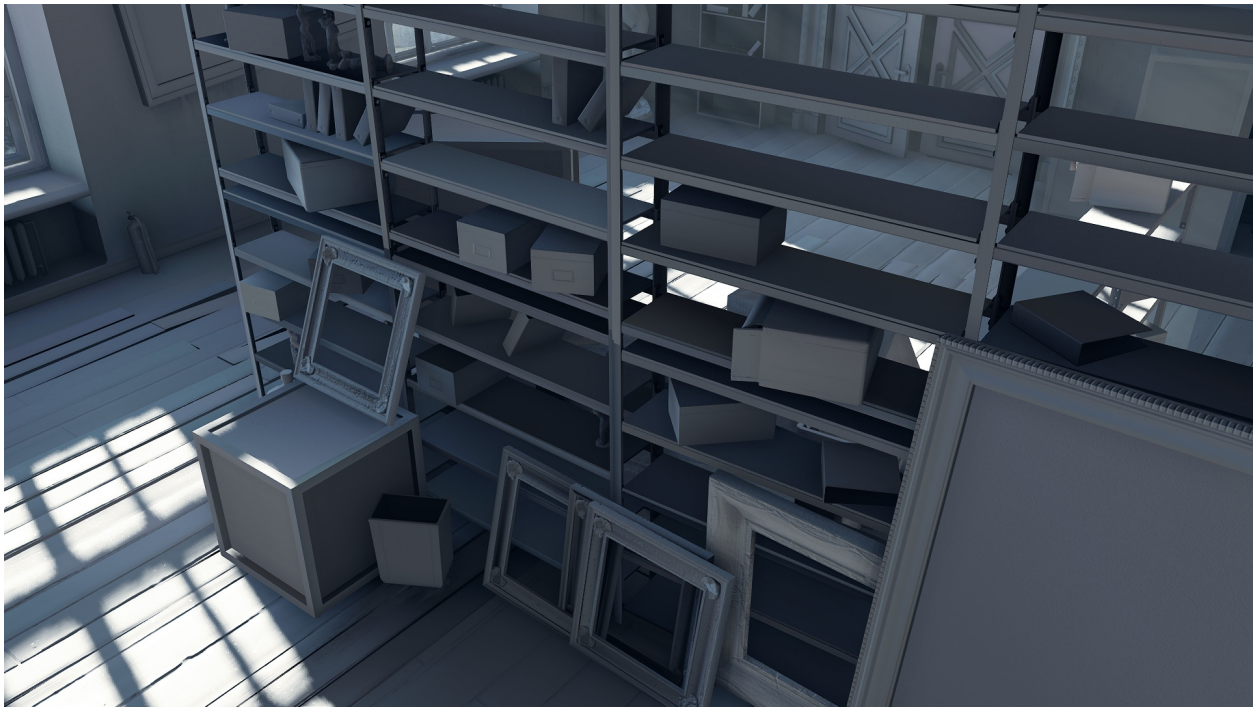

Рис. 13: До работы алгоритма

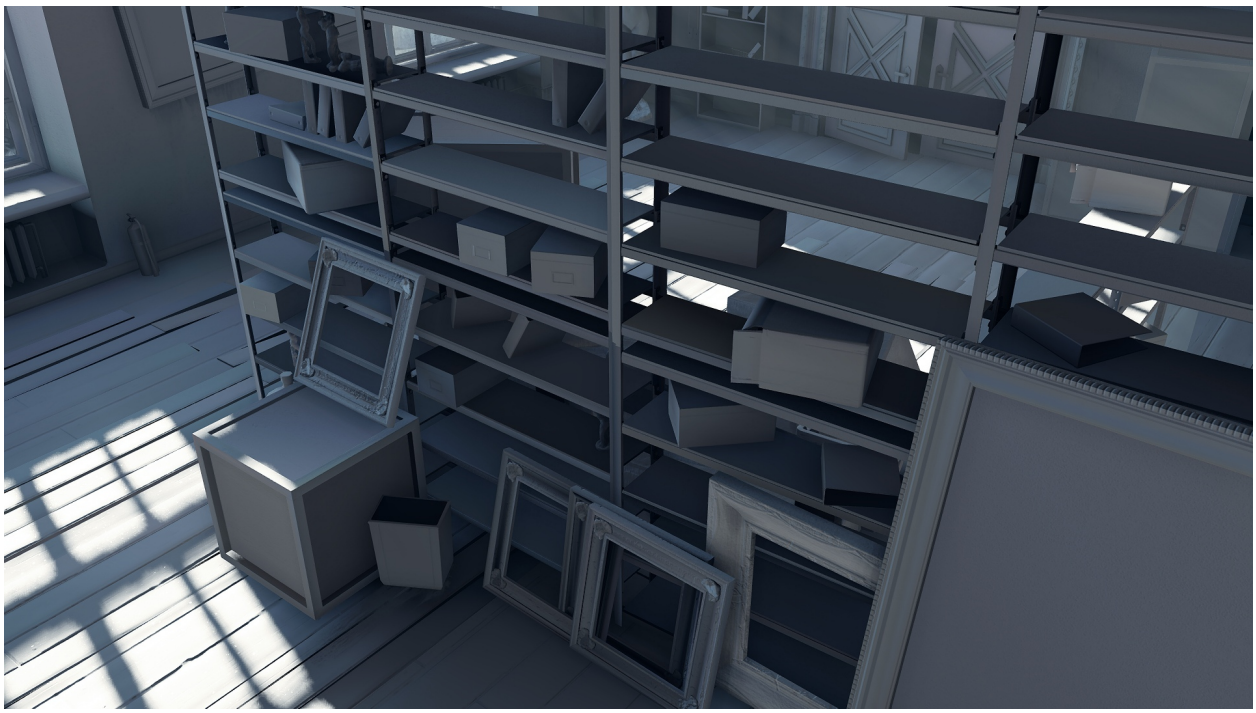

Рис. 14: После работы алгоритма

Заключение

По итогом данной работы:

- Была исследована реализация алгоритма lightmap в крупном игровом проекте World War Z.
- Был всесторонне исследован алгоритм lightmap, в том числе его реализация в других проектах.
- Пользуясь данными исследованиями было разработано несколько возможных оптимизацию текущего решения, для каждого из них была проведена серия тестов.
- Одна из оптимизаций была внедрена в проект, уменьшив размер lightmap примерно на 40 % и в сумме выиграв при этом порядка 300МБ.

Список литературы

[1] Microsoft. Texture block compression // docs.microsoft.com. — 2018. — URL: https://docs.microsoft.com/en-us/windows/desktop/direct3d11/texture-block-compression-in-direct3d-11 (online; accessed: 10.12.2018).

[2] Wikipedia. Kmeans // Википедия, свободная энциклопедия. — 2018. — URL: https://ru.wikipedia.org/wiki/K-means%2B%2B (дата обращения: 10.12.2018).

[3] Wikipedia. World War Z // Википедия, свободная энциклопедия. — 2018. — URL: https://ru.wikipedia.org/wiki/World_War_Z_(игра,_2019) (дата обращения: 10.12.2018).