ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

**Факультет Санкт-Петербургская школа
физико-математических и компьютерных наук**

Бубнов Данил Константинович

ИССЛЕДОВАНИЕ И ВНЕДРЕНИЕ АЛГОРИТМОВ АППАРАТНО-УСКОРЕННОГО СЖАТИЯ ТЕКСТУР В КАРТОГРАФИЧЕСКИЙ 3D-ДВИЖОК

Выпускная квалификационная работа - БАКАЛАВРСКАЯ РАБОТА
по направлению подготовки *01.03.02 Прикладная математика и информатика*
образовательная программа «Прикладная математика и информатика»

Рецензент
Разработчик,
ООО «ДГ-Софт»

_____
Айрих Владимир Александрович

Руководитель
к.ф. - м.н., доцент

_____
Мухин Михаил Сергеевич,

Консультант
Инженер-программист C++,
ООО «ДГ-Софт»

_____
Макаров Илья Олегович

Санкт-Петербург 2023

Оглавление

Аннотация

Картографические приложения сегодня стараются предоставить пользователям так называемый иммерсивный опыт. И приложение 2ГИС, которое рассматривается в рамках данной работы, также следует этому тренду. Для этого, например, используются детализированные модели зданий и их окружения. Для этих моделей обычно используются цветные текстуры высокого разрешения, которые требуют значительное количество оперативной памяти в процессе работы приложения, а также занимают значимое место в долговременной памяти (SD-карте на устройстве). Дополнительно к этому, значимое количество оперативной памяти требуют текстуры с текстами и иконками. А на мобильных устройствах оперативной памяти обычно мало, поэтому важно экономить её. Для уменьшения количества памяти, занимаемой текстурами, существуют специальные форматы сжатия. В данной работе рассмотрена возможность применения существующих форматов сжатия в картографическом трехмерном движке. Дополнительно, для текстур с текстовыми глифами был разработан алгоритм сжатия, который оказался существенно быстрее аналогов. Результатом этой работы стало то, что некоторые текстуры стали весить в 2 раза меньше, а некоторые в 4 или более раз. Благодаря этому у меньшего числа пользователей должны наблюдаться проблемы из-за недостатка оперативной памяти. Также это позволит дизайнерам добавлять большее количество детализированных моделей в приложение.

Ключевые слова: картографическое мобильное приложение, оперативная память, сжатие текстур

Map apps today are trying to provide users with what's called an immersive experience. And the 2GIS application, which is considered within the scope of this work, also follows this trend. For example, detailed models of buildings and their surroundings are used for this purpose. These models typically use high-resolution colored textures, which require a significant amount of RAM as the application runs, and also take up significant space in long-term memory (the SD card on the device). Additionally, textures with texts and icons require a significant amount of RAM. However, on mobile devices, RAM is usually scarce, so it's important to use less memory. There are special compression formats to reduce the amount of memory taken up by textures. This work considers the possibility of applying existing compression formats in a 3D map engine. Additionally, a compression algorithm was developed for textures with textual glyphs, which proved to be significantly faster than analogues. The result of this work was that some textures became two times lighter, and some four or more times lighter. Because of this, fewer users should have problems due to lack of RAM. It will also allow designers to add more detailed models to the application.

Keywords: map mobile application, RAM, texture compression

Введение

Описание предметной области

Картографические приложения в последнее время стараются предоставить пользователям иммерсивный опыт [3][11]. Приложение 2ГИС, с которым связана данная работа, также следует этому тренду. Например, в нем используются детализированные трехмерные модели зданий и их окружения. В будущем возможны ситуации, когда целые районы будут представлены такими детализированными моделями [6]. Для отображения деталей поверхности этих моделей (цвет, неровности и прочее) используются цветные текстуры (рис. 1). Текстуры используются также и для отображения других сущностей в приложении (рис. 2).

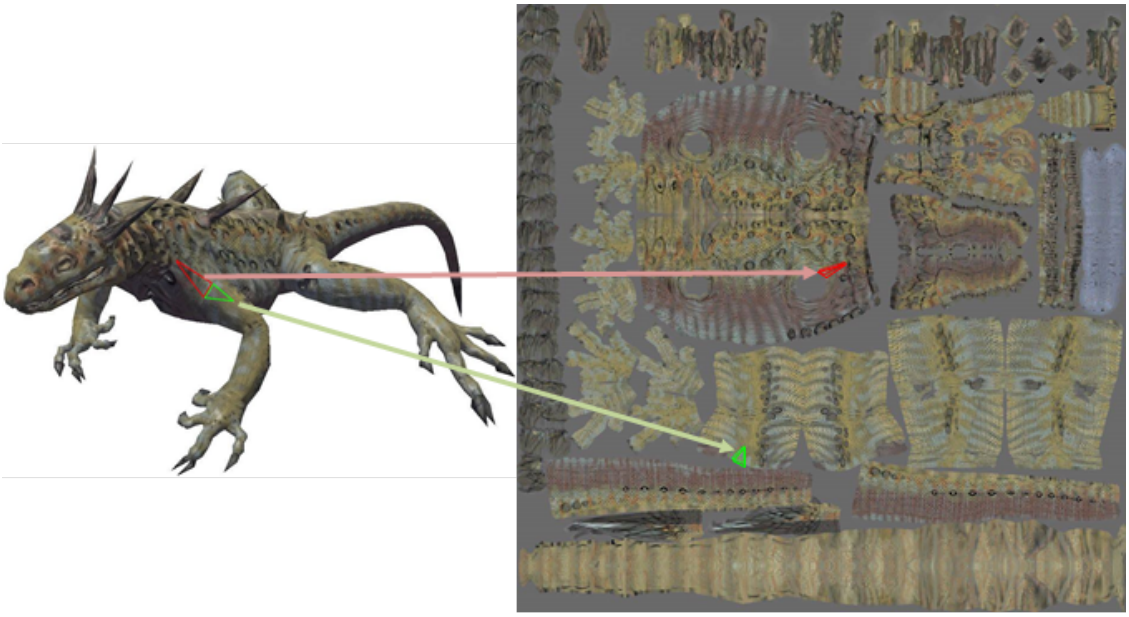
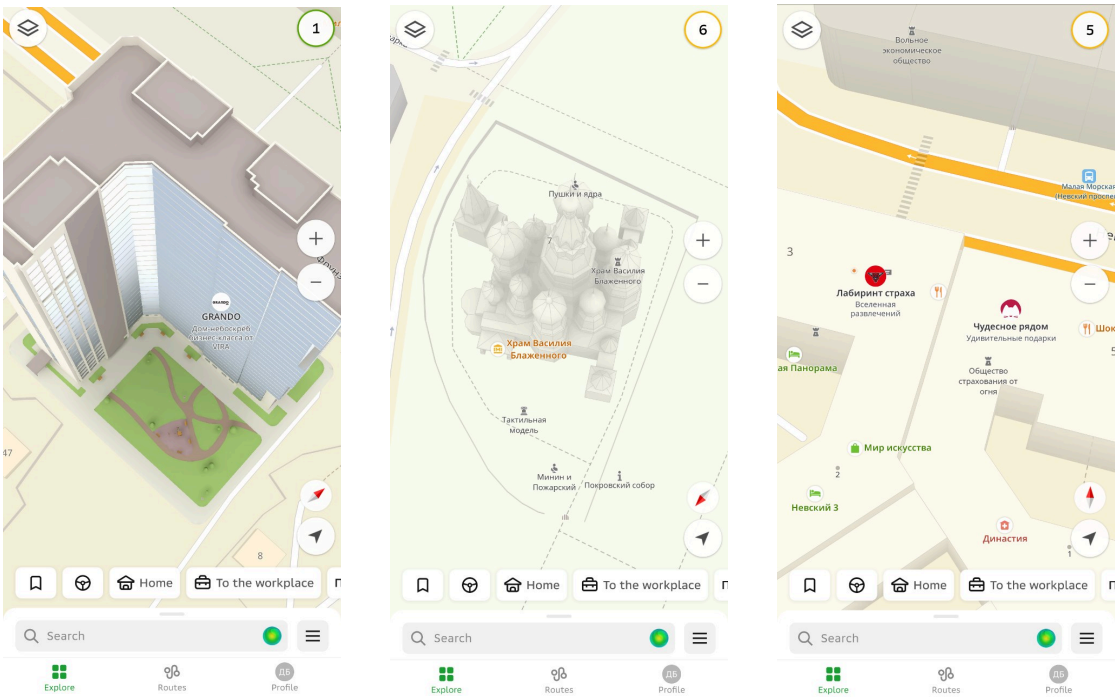


Рис. 1: Использование текстуры

В приложении 2ГИС можно выделить два основных типа текстур:

- Текстуры, генерируемые в процессе работы приложения (рис. 2c):
  - Текстуры с текстом — при попадании на экран объектов, которые должны быть подписаны (например, улицы или заведения), текст для них генерируется в текстуру прямо на



(a) Цветные модели (b) Черно-белые модели (c) Иконки и тексты

Рис. 2: Сущности, для которых используются текстуры

устройстве с помощью специальной библиотеки. При появлении новых объектов создается ещё одна текстура, а старая используется, пока виден хотя бы один объект, текст которого она содержит. За одну минуту может генерироваться больше тысячи текстур с текстом

– Текстуры с иконками компаний — логика генерации аналогична текстам, только генерируются из svg, которые предоставляют компании-рекламодатели. Генерируются ещё чаще, чем текстуры с текстами

– Текстуры с миниатюрами достопримечательностей и со стандартными иконками (по типу иконки парковки) — логика генерации аналогична текстам, только генерируются из svg, которые делают дизайнеры 2ГИС

- Текстуры, генерируемые заранее на сервере:

  – Цветные текстуры моделей (рис. 2a) — создаются дизайнерами в виде png или jpeg с помощью программ по типу Blender,

упаковываются вместе с моделью в специальный открытый формат glTF [10] и посылаются на устройство пользователя по сети. Таких моделей и текстур со временем будет только больше. У пользователя есть возможность скачать все данные города к себе на устройство, и тогда модели в этом городе будут браться из долговременной памяти устройства. В противном случае модель будет загружаться каждый раз заново, когда она понадобится (с точностью до кэширования)

– Черно-белые текстуры моделей (рис. 2b) — аналогично цветным, только используется свой закрытый формат, а не glTF. Новые текстуры для таких моделей вряд ли будут появляться

Рассмотрим процесс работы с текстурами, генерируемыми в процессе работы приложения (рис. 3). Из шрифта или SVG файла генерируется RG или RGBA изображение, в котором каждый пиксель занимает 2 или 4 байта. И это изображение копируется без изменений в память графического процессора и становится текстурой.

Рис. 3: Работа с генерируемыми текстурами

Теперь рассмотрим процесс работы с заранее сгенерированными текстурами (рис. 4). С сервера на устройство по сети посылается изображение в сжатом формате (JPG или PNG). Это изображение распакуется в RGBA формат, который весит в разы больше, чем исходный файл, и только после этого копируется в память графического процессора.

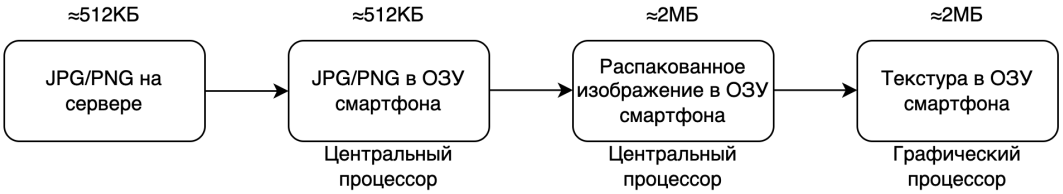


Рис. 4: Работа с заранее созданными текстурами

Проблема

После рассмотрения процесса работы с текстурами видно, что текстуры занимают в разы больше памяти, чем привычный нам форматы работы с изображениями (JPG и PNG). Замеры показали, что генерируемые текстуры в обычных сценариях работы с приложением (например, осмотр некоторого маршрута в Питере) занимают десятки мегабайт. Если моделей на экране много (а такие места на карте есть), то их текстуры также суммарно занимают десятки мегабайт.

Получается достаточно большое потребление оперативной памяти, которое негативно влияет в первую очередь на мобильную версию (есть ещё веб версия, которая не рассматривается в рамках данной работы) приложения, так как даже современные смартфоны сегодня могут иметь 2 или даже 1 гигабайт оперативной памяти. И большую часть оперативной памяти смартфона может занимать операционная система, а также другие приложения. Поэтому приложение при большом потреблении оперативной памяти может быть просто закрыто операционной системой. Но даже при достаточном количестве оперативной памяти её стоит экономить, так как в таком случае операционная система будет реже закрывать наше приложение, пока оно находится в фоне, то есть повторное открытие приложения будет происходить быстрее. А сценарий с частым сворачиванием и повторным открытием картографического приложения является довольно популярным, например, в режиме навигации по городу.

Также можно увидеть, что в процессе работы с текстурами на последнем шаге происходит копирование текстуры из памяти, которой

владеет центральный процессор, в память, которой владеет графический процессор. А память в смартфонах (особенно бюджетного сегмента) обычно медленная, то есть большой размер текстуры в ОЗУ может замедлять её появление на экране.

Поэтому хочется сократить использование оперативной памяти для текстур мобильным приложением. Для этого существуют специальные форматы сжатия текстур, которые в основном используются в игровых проектах [27][29]. Форматы отличаются друг от друга фиксированной степенью сжатия, качеством текстуры, которое можно получить в этом формате, и поддержкой целевыми устройствами. Для каждого формата существуют различные алгоритмы сжатия в виде утилит или библиотек. Они уже отличаются друг от друга скоростью сжатия и итоговым качеством текстуры.

Цель и задачи

Целью данной работы является уменьшение использования оперативной памяти мобильным приложением 2ГИС. Сжатие в 2 и более раз будет отличным результатом. Добиться этого планируется с помощью применения специальных форматов сжатия текстур. При этом хочется не сильно ухудшить важные для восприятия приложения характеристики: качество и размер файла для текстур, посылаемых по сети, и скорость генерации для текстур, создаваемых в процессе работы приложения.

Для достижения цели нужно исследовать существующие форматы и алгоритмы сжатия и выбрать наиболее перспективные. После этого для каждого типа текстур стоит рассмотреть выбранные форматы и алгоритмы и выбрать лучшие по важным для типа характеристикам.

Предварительная проверка результата будет проводиться в существующем тестовом приложении и с помощью уже существующих тестовых сценариев. Для него можно выбрать сценарии работы (например, пользователь не меняя масштаб исследует город) и проверить изменение характеристик приложения на этих сценариях. Качество изоб-

ражений будет сравниваться с помощью специальных метрик и с помощью опроса мнения группы людей (разработчиков и дизайнеров). Также, по возможности, стоит выбирать решения, которые можно будет в будущем интегрировать также и в веб версию приложения.

Ограничения

Стоит отметить, что свой формат сжатия создать в рамках диплома невозможно, так как поддержка формата должна быть на уровне железа или хотя бы графического программного интерфейса. Однако свой алгоритм сжатия в рамках существующего формата создать можно.

Структура работы

В главе 1 приведен обзор существующих форматов и алгоритмов сжатия, а также форматов доставки сжатых текстур. Также рассмотрено применение сжатия текстур в игровых проектах.

Глава 2 посвящена задаче о сжатии текстур с текстом.

Глава 3 посвящена задаче о сжатии текстур детализированных трехмерных моделей.

В главе 4 рассмотрен альтернативный способ подготовки трехмерных моделей, который позволяет лучше сжимать их текстуры.

1. Обзор литературы

1.1. Форматы сжатия

Обычные форматы сжатия изображений по типу png или jpeg не подходят для текстур, так как они не позволяют быстро обратиться к произвольному пикселю, а это важно для текстур. Поэтому существуют специальные форматы сжатия для текстур [14][29]. Все они сжимают текстуру с потерями, то есть итоговая текстура может отличаться от изначальной. Также все эти форматы являются блочными: текстура разбивается на блоки фиксированного размера, и каждый блок сжимается независимо в память фиксированного размера. Благодаря этому можно быстро обратиться к произвольному пикселю, разжав нужный блок. И это разжатие происходит автоматически на уровне графического API (OpenGL ES, Metal, Vulkan), то есть для разработчика работа с текстурой в шейдере не меняется.

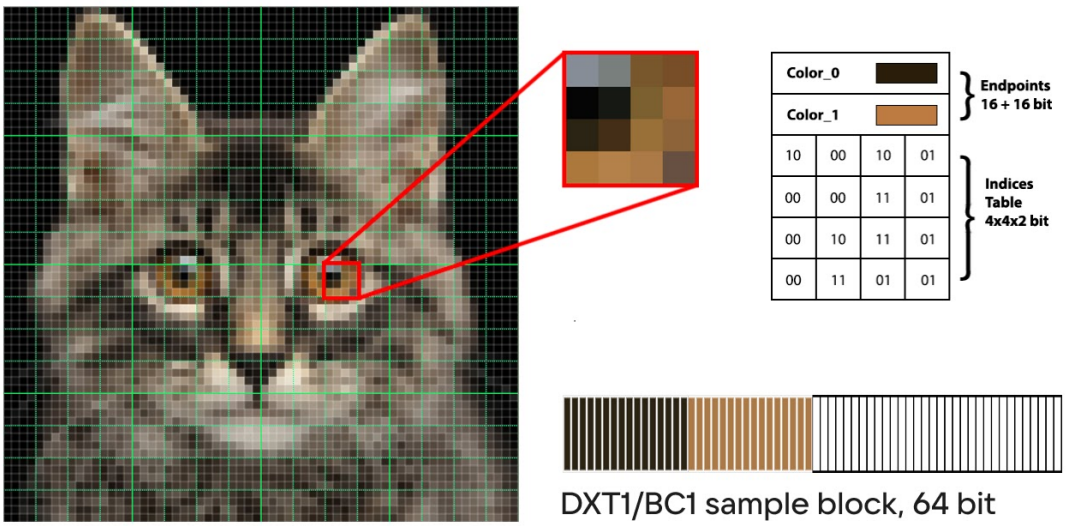


Рис. 5: Пример сжатия блока в формат BC1

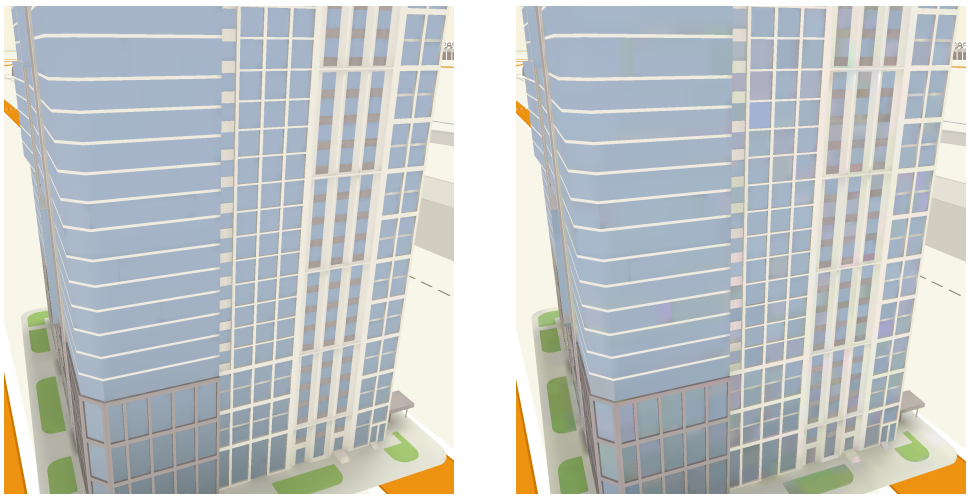
Последние данные от Apple [20] и Google [25] показывают, что на мобильных устройствах есть 4 распространенных формата сжатия: ETC1 [7], ETC2 [8], ASTC [2] и PVRTC [22]. Остальные (S3TC, ATC и прочие) поддержаны на небольшой доле мобильных устройств, поэтому они были исключены из рассмотрения. В виду того, что PVRTC поддержан

только на устройствах Apple и сравнения [23], которое показывает, что ASTC во всех аспектах (качество, наличие библиотек) лучше PVRTC, PVRTC был тоже исключен из рассмотрения. Также ETC2 является улучшением ETC1, поэтому нет смысла рассматривать ETC1. То есть в данной работе будут рассматриваться 2 формата: ETC2 и ASTC.

ETC2 имеет фиксированный размер блока (4 на 4 пикселя) и поддерживает сжатие текстур любого числа каналов (от 1 до 4). Для текстур с 1 или 2 каналами степень сжатия получается равной 2, для текстур с 3 каналами равной 6, для текстур с 4 каналами равной 4. Этот формат поддержан всеми устройствами, которые поддерживают OpenGL ES 3.0 или Metal любой версии. А приложение 2ГИС требует от целевого устройства поддержку OpenGL ES 3.1 (для Android) или Metal (для iOS), поэтому ETC2 поддержан на всех целевых устройствах 2ГИС.

ETC2 появился достаточно давно, поэтому для него есть несколько алгоритмов сжатия (в виде утилит или библиотек на языках C/C++). Например, есть утилиты etcpack [9] и etctool [12], которые выдают лучшее качество из возможных в этом формате, но работают достаточно долго (десятки секунд на MacBook Air M1). В качестве альтернативы есть утилита etcpak [26], которая работает намного быстрее всех аналогов, но при этом качество результата у неё хуже. Также была ещё одна попытка сделать быстрый алгоритм, который называется QuickETC [21], но его скорость сравнима с etcpak, и он не стал популярным.

ASTC является более новым форматом. Этот формат поддерживает различные размеры блока (от 4 на 4 до 12 на 12). От размера блока зависит степень сжатия (от 4 для блока 4 на 4 до 36 для блока 12 на 12) и качество итоговой текстуры (чем меньше блок, тем лучше качество). ASTC поддерживает сжатие только текстур с 4 каналами, но сжать текстуру с большим числом каналов можно, просто добавив ей каналов с произвольными значениями в них. Этот формат поддержан на всех устройствах, поддерживающих OpenGL ES 3.2 (для Android) или имеющих процессор Apple A8 или новее (для iOS). То есть он поддержан не на всех целевых устройствах, но по собранной с пользователей статистике можно сделать вывод, что он поддержан более чем 98 процентах



(a) Оригинальная текстура (b) Сжатая в ETC2 текстура

Рис. 6: Пример сжатия текстуры модели в ETC2

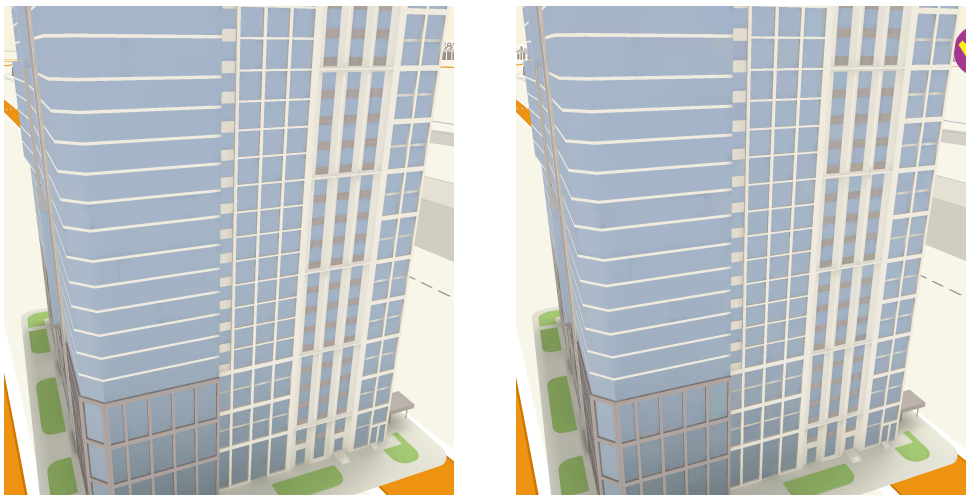
целевых устройств 2ГИС.

Для ASTC есть утилита astcenc [1] от создателей формата, которая позволяет гибко настраивать параметры сжатия. Например, можно выбирать качество итогового изображения, чем оно больше, тем медленнее будет работать сжатие. Также можно задать используемые в текстуре каналы, чтобы улучшить качество сжатия текстур с 1, 2 и 3 каналами.

Статья от Nvidia [24] показывает, что при использовании ASTC можно добиться лучшего качества, чем при использовании ETC2. Но при этом ASTC устроен сложнее, чем ETC2 (блок может делиться на регионы, и для каждого региона будут свои параметры). Это значит, что придумать свой алгоритм сжатия в ASTC будет сложнее.

Также есть форматы UASTC [28] (является подмножеством ASTC4x4) и ETC1S [31] (является подмножеством ETC1), которые позволяют перекодировать текстуру в любой другой формат в процессе работы приложения относительно быстро. То есть если использовать UASTC, то на устройстве, поддерживающем ASTC, можно очень быстро перекодировать текстуру в ASTC4x4, а на остальных устройствах в ETC2. Эти два формата также рассматривались в данной работе.

Для работы с этими форматами есть утилита basisu [4], которая позволяет гибко настраивать параметры сжатия. Например, там есть параметры, которые влияют на размер итогового файла, что может быть



(a) Оригинальная текстура (b) Сжатая в ASTC4x4 текстура

Рис. 7: Пример сжатия текстуры модели в ASTC

полезно, если мы посылаем текстуры по сети или храним на файловой системе устройства.

1.2. Форматы распространения

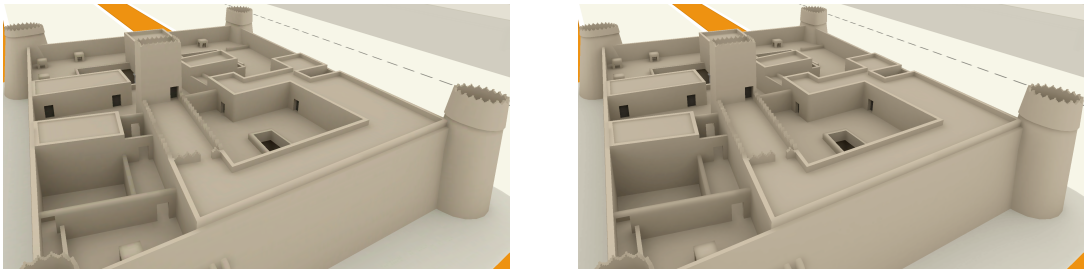
Сжатые текстуры можно доставлять до устройства в том виде, в котором они будет грузиться в память. Однако у такого подхода есть проблемы.

Во-первых, нужно будет как-то отличать форматы друг от друга, и нормально это делать получится только по расширению файла, что не очень хорошо, так как это повлечет за собой написание кода, который нужно будет постоянно поддерживать.

Во-вторых, файлы в таком виде весят достаточно много, так как форматы сжатия для текстур не являются самыми эффективными с точки зрения сжатия всей текстуры в целом.

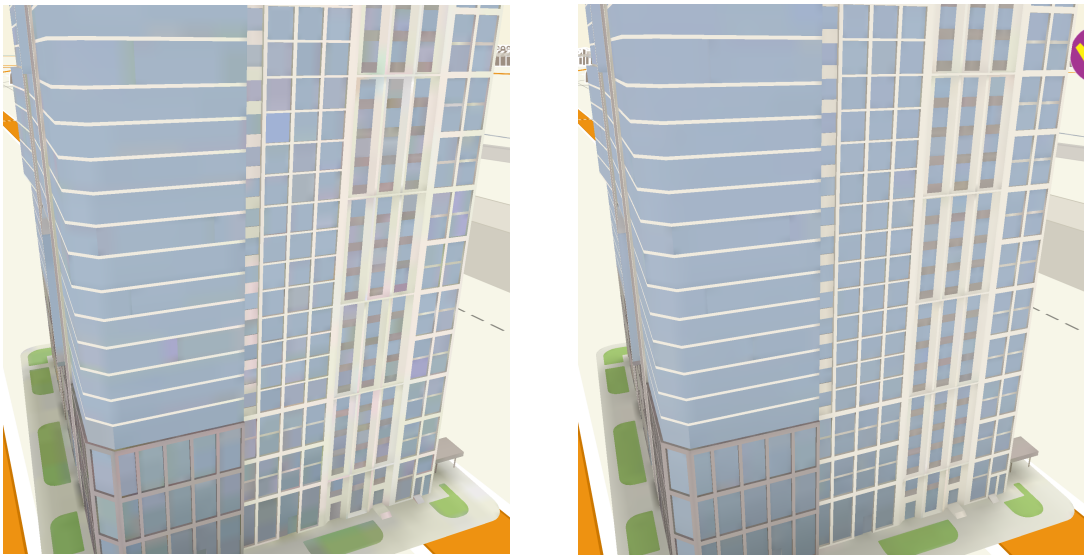
Для решения этих проблем существует формат KTX2 [5]. Он содержит внутри себя всю необходимую информацию про сжатую текстуру, которую можно получить при чтении с помощью библиотеки libktx [15]. Также он поддерживает сжатие своего содержимого без потерь с помощью обычных алгоритмов сжатия (например, Zstandard). Работать с KTX2 контейнерами можно с помощью утилит ktx2ktx2 и ktxsc [15].

1.3. Сравнение качества изображений



(a) Сжатие в ETC2. SSIM = 0.968 (b) Сжатие в ASTC4x4. SSIM = 0.999

Рис. 8: Пример значения метрики SSIM для сжатых текстур



(a) Сжатие в ETC2. SSIM = 0.998 (b) Сжатие в ASTC4x4. SSIM = 0.999

Рис. 9: Пример значения метрики SSIM для сжатых текстур

Есть популярные метрики для сравнения качества изображений: PSNR и SSIM [16]. Для сравнения текстур по этим метрикам можно использовать утилиты compare от imagemagick [18] или dssim [17]. В статье [30] эти метрики сравниваются, и из этой статьи можно сделать вывод, что SSIM обычно лучше чем PSNR отражает наличие заметных искажений в изображении. Однако не ясно, как понять, какие значения этих метрик являются приемлемыми, а какие нет.

В примере (рис. 8) значения SSIM отличаются, однако визуальных отличий на модели почти нет, командой, отвечающей за создание моделей, были бы одобрены оба варианта. Однако, в примере (рис. 9) значения почти не отличаются, но сжатие в ETC2 добавляет артефакты, и

такой вариант не был бы одобрен. Поэтому, если есть возможность, то качество лучше дополнительно проверять, спрашивая мнение группы людей (например, разработчиков и дизайнеров).

1.4. Применение сжатия в игровых проектах

Сжатие текстур широко используются в игровых проектах. Открытую информацию про использование сжатия можно найти в основном от движков: Unity, Unreal, Godot. Эти движки позволяют выбрать сжатие в зависимости от целевого устройства (на разные устройства придут разные текстуры) или одно для всех устройств. То есть предоставляют максимально вариативное решение. Однако, для данной работы такой подход подходит плохо.

Во-первых, у 2ГИС нет возможности делать разные версии приложения для разных устройств (можно добавить, но это будет очень трудоемко, так как требует вовлечения множества команд), а игровые движки такое поддерживают.

Во-вторых, в данной работе хочется получить решение, оптимизированное конкретно для приложения 2ГИС.

Также в приложении 2ГИС есть текстуры, генерируемые в процессе работы приложения, игровые же движки либо не поддерживают сжатие таких текстур, либо предлагают стандартное сжатие без попыток оптимизировать его. Ещё в приложении 2ГИС некоторые текстуры скачиваются по многу раз, а в играх обычно один раз во время установки.

2. Сжатие текстур с текстом

2.1. Работа

Первым делом работа велась над текстурами с текстом. Были собраны примерно 500 текстур с текстом, которые сгенерировало приложение за несколько минут работы. Эти текстуры получились разного разрешения, от 256 на 64 до 2048 на 1024 пикселей. Они двухканальные (1 канал цвета + альфа канал), то есть 1 пиксель занимает в оперативной памяти 2 байта. Для надписей на экране может быть одна или несколько текстур с текстом.

Такие текстуры генерируются довольно часто (до десятков в секунду), если пользователь перемещается по карте, поэтому, чтобы не было никаких задержек, сжатие должно быть максимально быстрым. Также, чем быстрее сжатие, тем меньше ресурсов процессора будет тратить приложение и батарея будет медленнее разряжаться. При этом сжатие должно происходить на одном потоке, так как распараллеливать, во-первых, непросто, а во-вторых, ядер в смартфонах обычно не так уж и много, и они заняты также другими задачами. Также текущая архитектура движка подразумевает 1 поток для подобных задач.

Сначала была попытка сжимать текстуры с текстом в ASTC формат с помощью библиотеки astcenc. Без доработок сжатие было достаточно медленным даже на самых быстрых настройках библиотеки. На слабых смартфонах сжатие крупной текстуры с тексом могло занимать до 300 миллисекунд, что более чем в 100 раз медленнее генерации текстуры без сжатия. И это является существенной задержкой, так как если нужно сжать несколько текстур, то они суммарно могли сжиматься больше секунды, и такое происходило часто. Вдобавок эта библиотека не умеет работать с 2 канальными текстурами, поэтому перед сжатием нужно выделять дополнительную память в два раза большего размера, что может наоборот увеличить потребление оперативной памяти в пике.

У текстур с текстом можно выделить 2 особенности: большую часть текстуры занимает либо белый цвет, либо черный, и часто встречаю-

щихся блоков (особенно 4 на 4 пикселя) должно быть не сильно много. Учитывая эти особенности, была предпринята попытка ускорить сжатие с помощью кэша: если блок уже сжимался, то для него можно сразу же вернуть результат. Это сильно ускорило сжатие, но хотелось ещё быстрее. Также у версии с ASTC кэш был не идеальный: в качестве ключа нужно использовать что-либо содержащее 16 байт, а значит кэш должен быть в виде хэш таблицы, а не массива. И ещё никуда не пропала проблема с необходимостью выделения дополнительной памяти для сжатия.

Поэтому дальше работа велась над алгоритмы сжатия в ETC2. Как уже было сказано выше, для сжатия в ETC2 есть утилита etcpak, которая является самой быстрой утилитой для сжатия. Она была уже в 3 раза быстрее astcenc. Однако она не подразумевает её использование в качестве библиотеки: нет CMakeLists, нет документации. Вдобавок эта утилита также не умеет работать с двухканальными текстурами, поэтому здесь есть та же проблема с выделением дополнительной памяти.

В итоге было решено попробовать придумать новый алгоритм в надежде получить ещё более быстрое решение, лишенное проблем аналогов. В формате ETC2 для сжатия одного канала используется блок EAC (рис. 10). В нем для всех пикселей блока выбирается единый базовый цвет (Base), множитель (Mul) и индекс строки в предопределенной таблице (N). И для каждого пикселя индивидульно выбирается индекс в выбранной строке, который занимает 3 бита. И используя эти 4 компонента по формуле получается цвет пикселя.

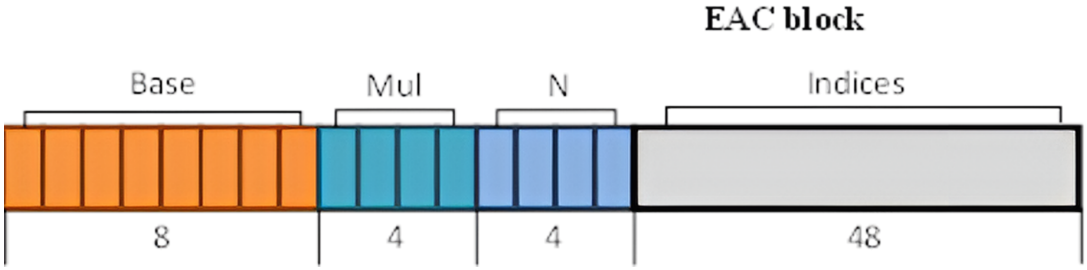


Рис. 10: Устройтсво блока EAC

И при сжатии текстур с текстом можно независимо сжимать оба канала в этот формат. Вдобавок, в этом формате общие для пиксе-

лей в блоке параметры (базовый цвет, множитель и индекс строки) не зависят от порядка цветов в блоке. Поэтому можно попробовать предподсчитать значения этих параметров для каких-нибудь блоков. Также было предположение, что градации цветов в текстурах с текстом не очень важны, так как большую часть текстуры занимают белый и черный цвета.

Был придуман следующий алгоритм сжатия блока текстуры:

0. Общие для пикселей в блоке параметры предподсчитываются полным перебором для всех блоков, состоящих из значений 0, 32, 64, 96, 128, 160, 192, 224 и 255. При этом блоки без 0 (черный) и 255 (белый) не рассматриваются. Таких блоков получается $2^7 = 128$, так как порядок пикселей в блоке не важен. Ключом, по которому можно будет получить посчитанные значения, является битовая маска цветов в блоке (наличие 32 - первый бит, наличие 64 - второй бит и т.д.) (число от 0 до 127).

1. Числа в блоке приводятся к ближайшему кратному 32 числу

2. Получается множество новых чисел в блоке

3. Это множество кодируется в битовую маску – получается ключ

4. По этому ключу получаются предподсчитанные параметры

5. Для каждого исходного пикселя выбирается оптимальный индекс (из 8) в строке, определенной полученными параметрами

6. Полученные значения пакуются в нужном виде в EAC

Сравнение полученного алгоритма с аналогами (Скорость = Скорость сжатия 500 текстур с текстом различного размера на одном потоке MacBook Air M1, W = ширина, H = высота):

Рис. 11: Иллюстрация к шагам 1-3

	etcpak	astcenc	Новый алгоритм
Скрость	≈980мс	≈2917мс	≈120мс
Степень сжатия	2	2	2
Есть библиотека	-	+	+
Доп память	$\mathcal{O}(W \cdot H)$	$\mathcal{O}(W \cdot H)$	$\mathcal{O}(W)$

При реализации в движке сжатие происходит прямо в исходную память текстуры с дополнительным выделением $\mathcal{O}$(ширина текстуры) памяти, в то время как аналоги не умеют сжимать в ту же память, и, как было сказано выше, требуют на вход текстуру с 4 каналами, на что тоже нужно много дополнительной памяти.

Полученный алгоритм оказался во всём лучше аналогов, поэтому было решено остановиться на нём.

2.2. Тестирование

Тестирование производительности проводилось с помощью существующих бенчмарков 2ГИС. Они прогоняют приложение на смартфоне Samsung A8 (средний смартфон 2018 года) в различных сценариях (облет между объектами, облеты со сменой масштаба и т.д.) и замеряют время до первой отрисовки кадра, частоту кадров, загруженность процессора и другие. Среднее время до первой отрисовки кадра увеличилось примерно на 8 миллисекунд (было 80, стало 88), что ожидаемо, так как текстура с текстом по проведенным отдельно замерам на слабом смартфоне сжимается не больше пары миллисекунд. Средняя частота

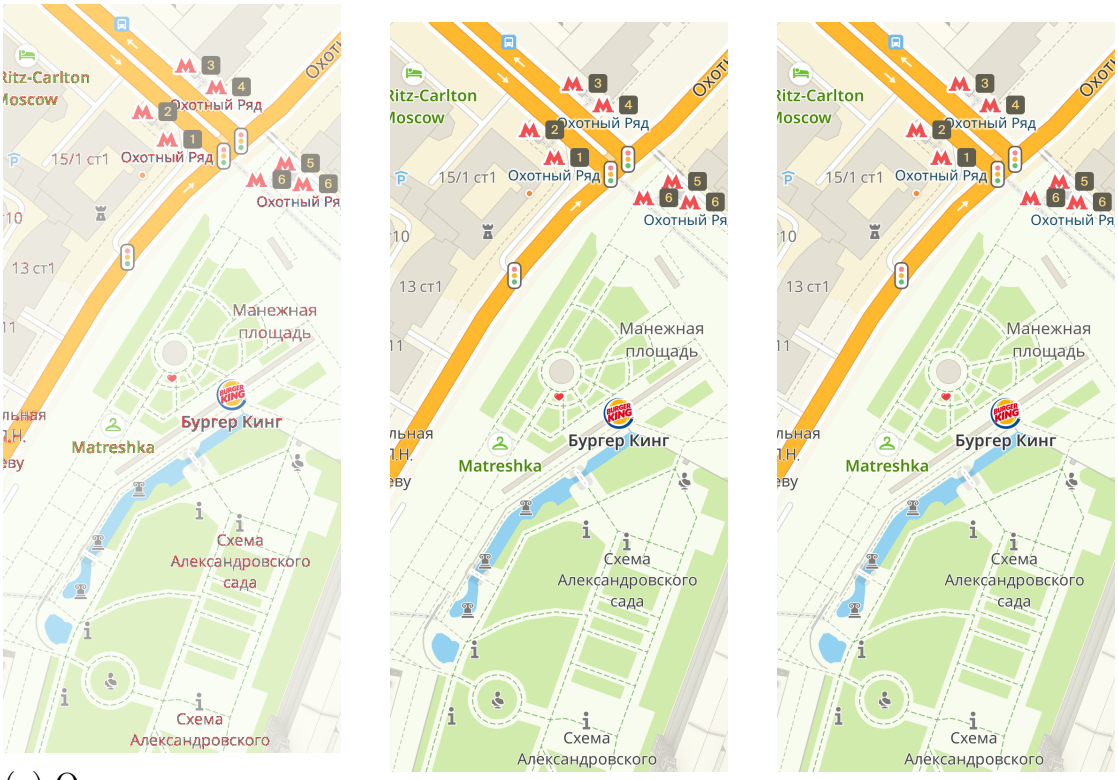
кадров в худшем случае упала примерно на 3 кадра (было 27, стало 24). На это могло повлиять медленное разжатие блоков. Возможно, что в старых смартфонах оно делается не очень эффективно. Также возможно, что это неточность теста, и из-за замедления отрисовки первого кадра уменьшается и частота кадров в статистике. Загрузка процессора в худшем случае поднялась на 3 процентных пункта, что отчасти можно списать на погрешность теста, так как в автоматических запусках этого теста загрузка процессора тоже может немного отличаться. В целом, результаты оказались ожидаемыми и удовлетворительными, сжатие можно использовать.

Тестирование визуального качества также проводились с помощью существующих тестов 2ГИС. Они отрисовывают некоторые участки карты и попиксельно сравнивают изображения ”до” и ”после”. И итоговые изображения было также предложено сравнить членам команды, они не заметили разницы в них, удобочитаемость текста не изменилась. Можно было бы сравнить итоговые изображения с помощью метрик SSIM и PSNR, но как было отмечено выше, нельзя сказать, при каких значениях этих метрик становятся видны изменения.

Тестирование памяти также проводилось с помощью предыдущего теста, в нем в отдельных сценариях экономия достигала до 7 мегабайт (было 26.66, стало 19.60).

2.3. Выводы

Для сжатия текстур с текстами в приложении 2ГИС был разработан новый алгоритм на основе предподсчета, который оказался примерно в 8 раз быстрее самого быстрого аналога. Тестирование показало, что негативное влияние на приложение от сжатия не сильное, и при этом оперативной памяти экономится до 7 мегабайт, что достаточно хорошо. Поэтому данный алгоритм был внедрен в приложение 2ГИС.



(a) Отличающиеся пиксели помечены красным

(b) Без сжатия

(c) Со сжатием

Рис. 12: Пример изображений из теста

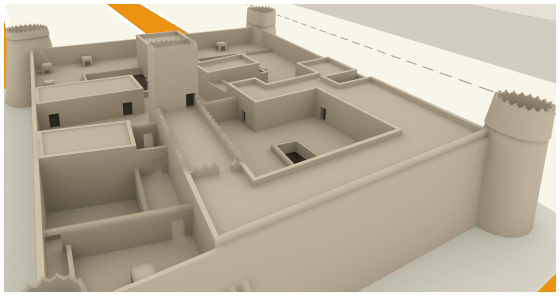
3. Сжатие текстур моделей

3.1. Работа

В отличие от текстов, текстуры моделей передаются по сети, а также, возможно, входят в пакет города, который можно скачать один раз и сохранить на диск. Поэтому их можно сжать заранее, а значит скорость сжатия уже не является важным критерием. Для них теперь важен размер файла, чтобы быстрее скачивать, тратить меньше трафика и занимать меньше места на диске. У любого формата степень сжатия будет как минимум 4, что уже очень хорошо. И если можно увеличить степень сжатия, но при этом изменение качества станет заметным, то такой вариант будет отвергнут ответственными за создание моделей.

У моделей 2ГИС размеры текстур бывают разные. Есть большого разрешения (1024 на 1024 пикселя) и есть небольшого (256 на 256 пикселей). Также есть более простые текстуры (рис. 13а, в них меньше

деталей и резких переходов цветов) и более сложные (рис. 13b, в них много деталей, может быть зеркальность и отражения), а также что-то среднее между ними.



(a) Текстура с малым числом деталей

(b) Текстура с большим числом деталей

Рис. 13: Пример моделей с текстурами

На первый взгляд казалось, что для такой задачи как раз подойдут универсальные форматы UASTC и ETC1S. Для работы с этими форматами была использована утилита basisu. На существующих моделях ETC1S обычно имеет плохое качество (рис. 14) как по метрикам, так и по мнению команды. Но на простых моделях (рис. 13a) он может подойти по качеству, и тогда можно его использовать, так как размер файла в этом формате получается минимальным.

UASTC же обладает хорошим качеством, так как основан на ASTC4x4, однако файлы в таком формате имеют достаточно большой размер по сравнению с использующимся сейчас jpg. В утилите basisu для работы с UASTC есть параметры uastc_rdo_l и uastc_rdo_d, которые позволяют уменьшить размер итогового файла, однако уменьшение получается не сильное, и при этом падает качество текстуры. Если же размер файла будет устраивать команду, то его можно использовать, так как он выдает отличное качество и поддержан на всех целевых устройствах.

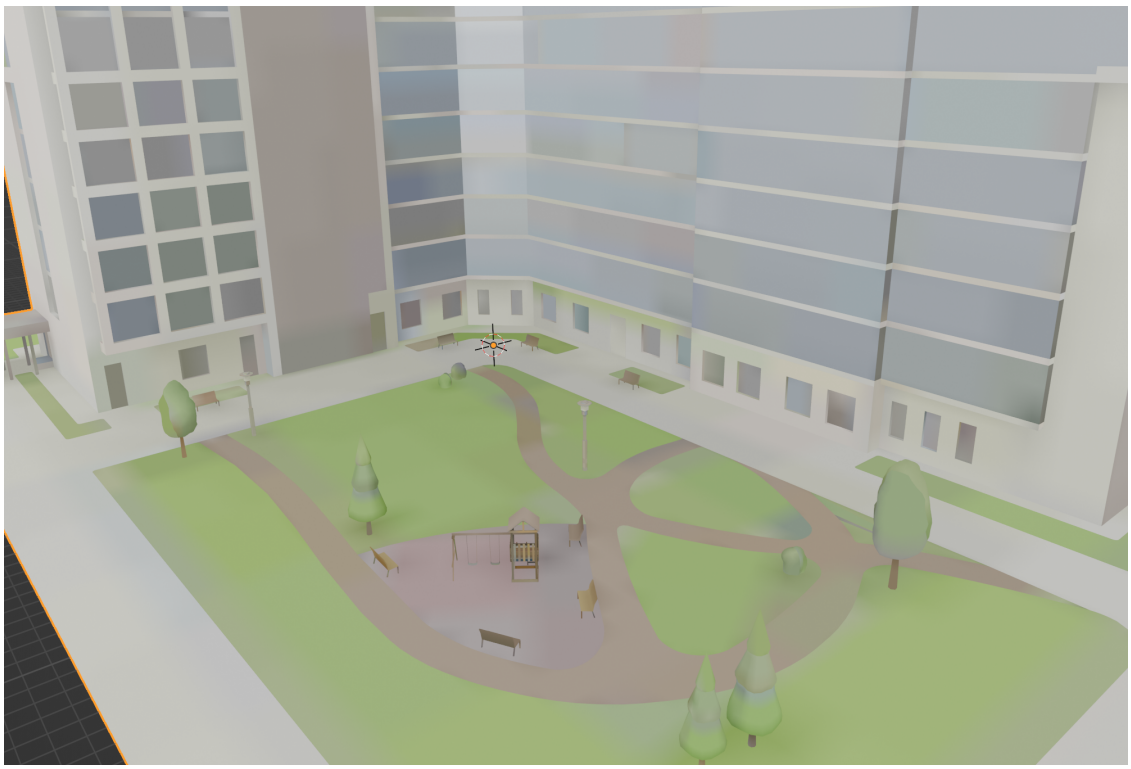


Рис. 14: Пример плохого сжатия в ETC1S: заметны артефакты сжатия

Так как универсальные форматы оказались не идеальными, были рассмотрены также ETC2 и ASTC. Для работы с ETC2 были использованы утилиты etcpack и etctool, так как они дают наилучшее качество. Из преимуществ данного формата для этой задачи можно выделить степень сжатия равную 8, малый размер итогового файла и поддержку всеми целевыми устройствами. Однако качество по метрикам и на глаз не всегда получается хорошим. Для простых текстур (рис. 8а) качество может быть приемлемым, а на более сложных текстурах или текстурах с меньшим разрешением могут быть заметны артефакты (рис. 9а).

Для работы с ASTC была использована утилита astcenc, так как у неё также лучшее качество. У ASTC можно выбирать размер блока. При увеличении размера блока увеличивается степень сжатия (а соответственно и уменьшается размер файла), но ухудшается качество. У ASTC4x4 почти всегда получается идеальное качество, неотличимое от оригинала, но как и у UASTC есть проблема с размером файла. Поэтому были также испытаны увеличенные размеры блока, при которых бы размер файла стал приемлемым и при этом качество не сильно ухудши-

лось. Иногда при размере блока 6x6 получалось приемлемое качество и хороший размер файла. Иногда же при таком размере блока качество было уже заметно хуже оригинала и приходилось брать, например, 5x5. При этом степень сжатия будет больше, чем у ETC2 только при размере блока 6x6 или больше. Также у ASTC есть недостаток: он поддержан не на всех целевых устройствах (однако поддержка более 98 процентов и со временем будет только расти). Поэтому на устройствах, которые не поддерживают этот формат, нужно придумать механизм работы с ними. Можно либо разжимать текстуру (то есть не будет сжатия) с помощью astcenc (в виде библиотеки), либо вовсе на таких устройствах не показывать такие модели.

То есть в итоге для текстур моделей не удалось выбрать универсальный идеальный вариант для всего. Поэтому было решено, что для каждой текстуры модели сжатие будет выбираться индивидуально. В движок была добавлена поддержка всех рассмотренных форматов с сохранением обратной совместимости (модели без сжатия продолжат работать как раньше). Для ASTC было решено разжимать текстуру, если устройство не поддерживает этот формат. Замеры показали, что такое разжатие работает быстрее разжатия jpg, поэтому проблем со скоростью загрузки текстуры в таком случае не будет.

3.2. Доставка сжатых текстур

Для хранения и доставки трехмерных моделей в 2ГИС используется открытый формат glTF. Однако, он по умолчанию не поддерживает хранение сжатых текстур (то есть библиотеки для работы с glTF не поддерживают, в частности используемая в 2ГИС). Но для UASTC и ETC1S было создано расширение KHR_texture_basisu [13] для glTF, которое добавляет поддержку сжатых текстур в описанном в обзоре литературы контейнере ktx2. Сам контейнер ktx2 умеет хранить текстуру в любом сжатом формате. И хочется иметь поддержку хранения в glTF ktx2 контейнера с любым форматом сжатия внутри.

Движок 2ГИС использует библиотеку tinygltf, и она позволяет по-

лучить сырые данные текстуры, а также mime тип текстуры. Поэтому реализация в движке читает этот mime тип, и если он равен image/ktx2, то с помощью библиотеки libktx извлекает из контейнера сжатую текстуру. Осталось научиться нужным образом упаковывать ktx2 в glb файл (бинарный подвид glTF). Для этого использовалась утилита gltf-transform [19], она позволяет распаковать glb файл, после этого можно подменить текстуры на ktx2 контейнеры и изменить их mime тип, и в конце запаковать всё обратно. То есть в итоге оказалось возможным добавить поддержку хранения в glTF ktx2 контейнера с любым форматом сжатия внутри.

Также были проведены тесты скорости разжатия ktx2 контейнера и сжатой текстуры в разжатую. Оказалось, что разжатие ktx2 контейнера на порядок быстрее разжатия jpg. И разжатие сжатой текстуры также быстрее разжатия jpg. То есть новый подход даже увеличит скорость чтения текстур, что может уменьшить задержку появления модели на экране.

Сценарий	Среднее время
Распаковка jpg	≈34мс
Распаковка ktx2 с ETC2	≈1.7мс
Распаковка ktx2 с ASTC5x5	≈1.3мс
Распаковка ktx2 с ASTC5x5 + разжатие в RGBA32	≈25мс
Распаковка ktx2 с UASTC + перевод в ASTC4x4	≈25мс
Распаковка ktx2 с UASTC + перевод в ETC2	≈47мс

3.3. Решение для создателей моделей

Для создателей моделей был создан Docker образ со всеми нужными утилитами, а также скрипт, принимающий glb файл и выдающий по glb файлу со сжатыми текстурами для каждого формата. После этого все эти модели можно открыть в движке и посмотреть на все разом. Рекомендуется рассматривать форматы в порядке: ETC1S -> ETC2 -> UASTC -> ASTC12x12 -> ... -> ASTC4x4. Можно использовать первый формат, качество которого будет устраивать создателей моделей.

3.4. Выводы

Для текстур моделей выбрать единый формат сжатия не удалось. У каждого формата есть свои преимущества и недостатки. Поэтому в движок была добавлена поддержка всех рассмотренных форматов сжатия. Создатели моделей с помощью предоставленного им инструментария будут выбирать оптимальный формат сжатия для каждой модели индивидуально. Коэффициент сжатия в любом из форматов будет не меньше 4, это позволит в сложных сценариях (район из детализированных моделей) экономить десятки мегабайт.

4. Другой подход к созданию текстур моделей

4.1. Работа

Основной причиной, из-за которой не удалось выбрать единый формат сжатия для моделей, являлось то, что текстуры бывают разными: детализированными и простыми. Простые текстуры хорошо выглядят почти в любом сжатом формате, в отличие от детализированных текстур. Чтобы улучшить сжатие можно попробовать изменить сами текстуры, чтобы они стали проще.

Сейчас процесс создания большинства текстур моделей происходит следующим образом: сначала граням модели задается цвет, а затем ставится освещение, тени от которого вместе с цветом запекаются в текстуру (рис. 15). Но такая текстура является трехканальной (а движок добавляет ещё альфа канал и заполняет его стандартным значением), и если в ней много цветов, то она получается сложной для сжатия.

Но можно было бы запекать затенение в одноканальную текстуру, а цвета хранить для граней, как на первом этапе создания моделей. Затем в приложении на этапе отрисовки цвет будет умножаться на величину затенения, и результат должен быть не отличим от подхода с трехканальной текстурой. Информация про цвета на гранях весит сильно меньше текстур, поэтому экономия по памяти будет примерно четырехкратная даже без сжатия из-за меньшего числа каналов.

И самое главное – одноканальная текстура является простой (в ней не больше 256 цветов), и для её сжатия можно использовать формат EAC, который был использован при сжатии текстур с текстом. Этот формат дает отличное качество и сжимает текстуру в 2 раза. А также он поддержан на всех целевых устройствах. Другие форматы в данном случае во всем хуже EAC. Кроме экономии оперативной памяти, уменьшится также размер файла с текстурой. Также, при том же разрешении текстуры видно меньше артефактов, чем при использовании обычной текстуры и сжатия её в jpg.

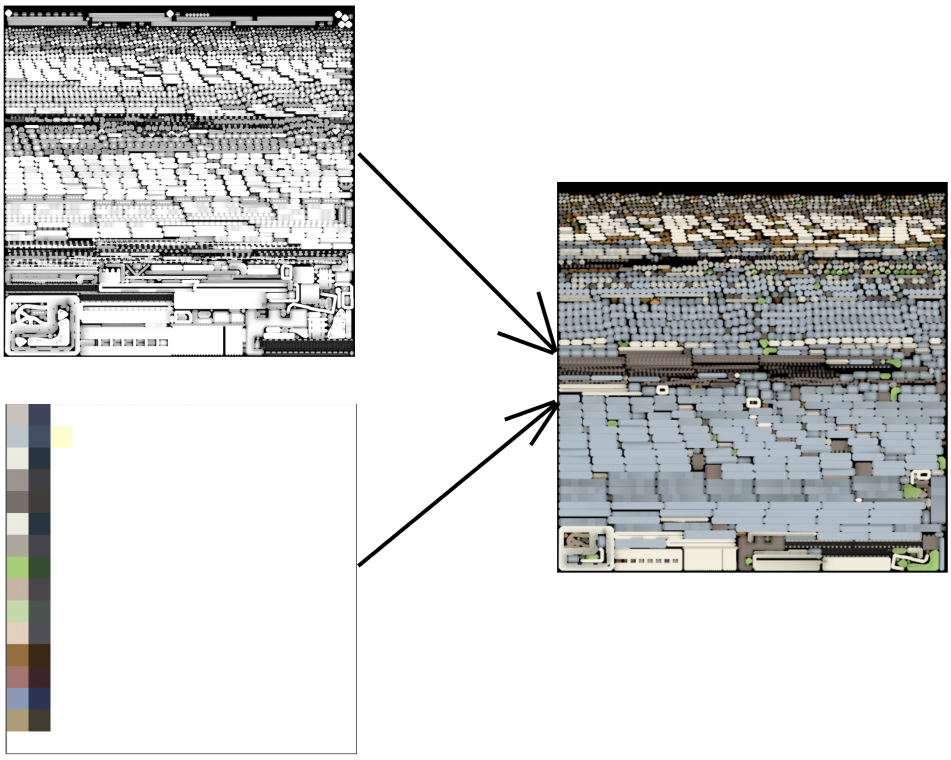


Рис. 15: Текущий подход к созданию текстуры

Однако есть модели, в которых запекается также освещение от окон или зеркальность (рис. 13b). Возможно, текстуры таких моделей не получится изменить предложенным образом. Поэтому предыдущий блок про сжатие текстур моделей остается актуальным.

4.2. Экспорт модели из Blender

В рамках данной задачи в движок была добавлена поддержка текстур с затенением для моделей с сохранение обратной совместимости (модели без затенения будут работать как раньше). Также была написана статья для создателей моделей, описывающая как нужно экспортировать модель с затенением в отдельной текстуре, чтобы движок смог её прочитать.

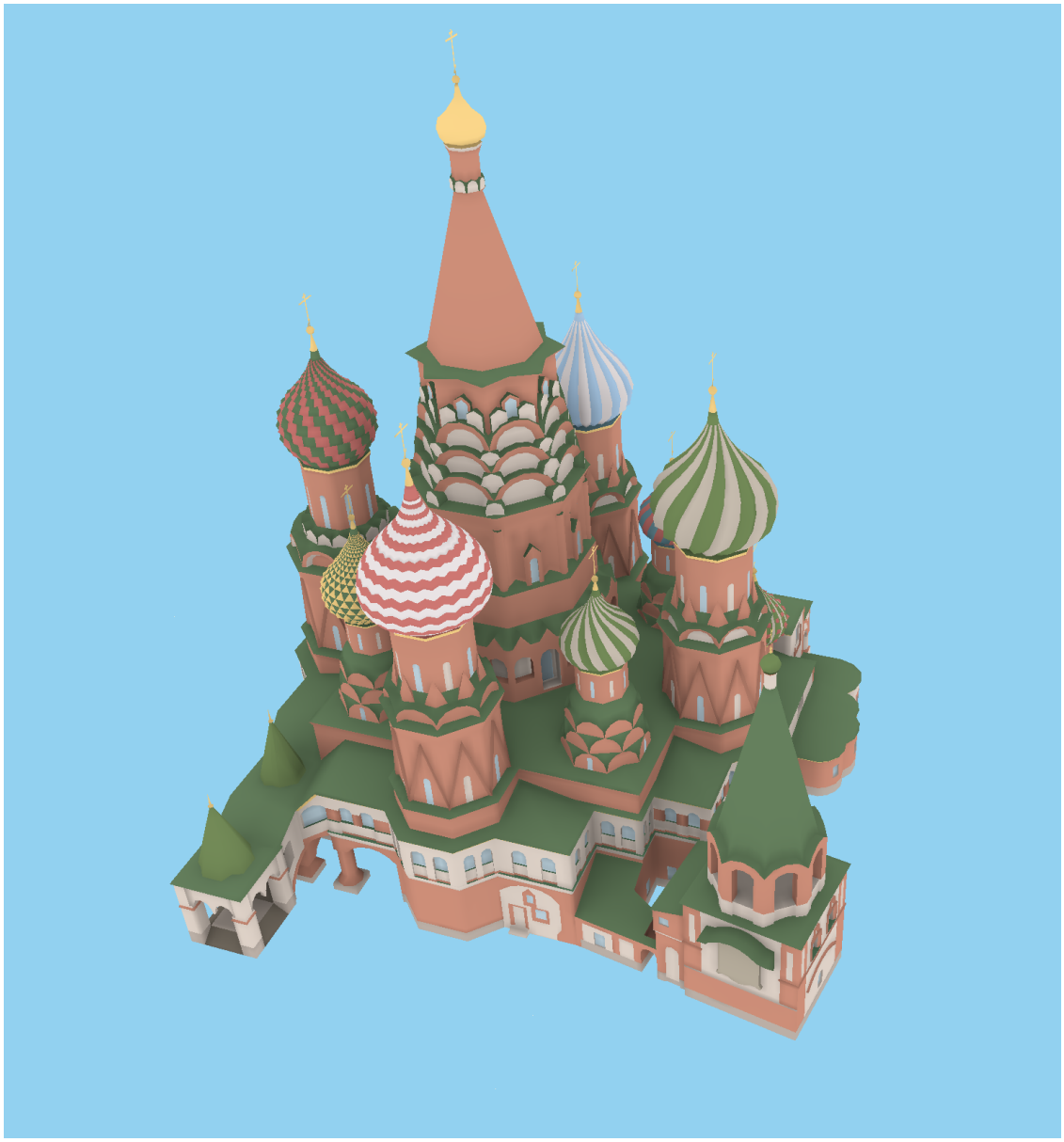


Рис. 16: Пример модели с использованием нового подхода

4.3. Выводы

Для улучшения сжатия текстур моделей было предложено изменить сами текстуры, а именно – вместо цветной текстуры с запеченным в неё затенением хранить затенение в одноканальной текстуре и цвета на гранях. Такой подход упрощает текстуры, и их становится легче сжимать. Однако, не для каждой модели можно применить этот подход. Также осталось протестировать этот подход: влияет ли смешение цвета с затенением при отрисовке на производительность приложения.

Заключение

В данной работе были исследованы и применены форматы сжатия текстур для мобильного приложения 2ГИС. В результате текстуры в самых нагруженных сценах стали занимать в оперативной памяти на десятки мегабайт меньше.

Для текстур с текстом был разработан и внедрен в движок новый алгоритм сжатия в формат EAC, который оказался во всём лучше аналогов. Коэффициент сжатия для этих текстур получился равным двум.

Для текстур моделей выбрать единый формат сжатия не удалось, у каждого были свои преимущества и недостатки. Поэтому создателям моделей был предоставлен удобный способ сжимать текстуры в различные форматы, чтобы они могли выбрать оптимальный формат для каждой конкретной модели. Также в движок была добавлена поддержка всех изученных форматов сжатия. Коэффициент сжатия для этих текстур получился не меньше четырех.

Дополнительно был предложен и исследован измененный подход к созданию некоторых моделей, в котором затенение запекается в отдельную текстуру и уже в движке смешивается с цветом граней модели. При таком подходе получается экономия двух каналов текстуры, и вдобавок новую текстуру можно сжимать в EAC формат, дающий отличное качество и степень сжатия. Коэффициент сжатия для этих текстур получился равным восьми.

В будущем нужно будет рассмотреть сжатие оставшихся типов текстур: текстур с иконками компаний и текстур с обычными иконками.

Список литературы

[1] ARM-software. astc-encoder. — 2022. — URL: https://github.com/ARM-software/astc-encoder (дата обращения: 17.05.2023).

[2] ASTC Compressed Texture Image Formats. — The Khronos Group Inc., 2020. — URL: https://clck.ru/34VUjq (дата обращения: 17.05.2023).

[3] Apple Maps introduces new ways to explore major cities in 3D. — Apple, 2021. — URL: https://www.apple.com/newsroom/2021/09/apple-maps-introduces-new-ways-to-explore-major-cities-in-3d/ (дата обращения: 17.05.2023).

[4] BinomialLLC. basis_universal. — 2022. — URL: https://github.com/BinomialLLC/basis_universal (дата обращения: 17.05.2023).

[5] Callow M. KTX File Format Specification. — 2022. — URL: https://registry.khronos.org/KTX/specs/2.0/ktxspec.v2.html (дата обращения: 17.05.2023).

[6] Create immersive 3D map experiences with Photorealistic 3D Tiles. — Goole, 2023. — URL: https://cloud.google.com/blog/products/maps-platform/create-immersive-3d-map-experiences-photorealistic-3d-tiles/ (дата обращения: 17.05.2023).

[7] ETC1 Compressed Texture Image Formats. — The Khronos Group Inc., 2020. — URL: https://clck.ru/34VUgq (дата обращения: 17.05.2023).

[8] ETC2 Compressed Texture Image Formats. — The Khronos Group Inc., 2020. — URL: https://clck.ru/34VUhm (дата обращения: 17.05.2023).

[9] Ericsson. ETCPACK. — 2016. — URL: https://github.com/Ericsson/ETCPACK (дата обращения: 17.05.2023).

[10] GPU pro 5: Advanced rendering techniques / Fabrice Robinet, Remi Arnaud, Tony Parisi, Patrick Cozzi. — CRC Press, 2014. — C. 375–392.

[11] Immersive view coming soon to Maps. — Goole, 2022. — URL: https://www.blog.google/products/maps/three-maps-updates-io-2022/ (дата обращения: 17.05.2023).

[12] Inc. Google, Inc. Blue Shift. Etc2comp. — 2017. — URL: https://github.com/google/etc2comp (дата обращения: 17.05.2023).

[13] KHR_texture_basisu. — KhronosGroup, 2021. — URL: https://github.com/KhronosGroup/glTF/blob/main/extensions/2.0/Khronos/KHR_texture_basisu/README.md (дата обращения: 17.05.2023).

[14] Khronos Data Format Specification V1.3.1. — The Khronos Group Inc., 2020. — URL: https://registry.khronos.org/DataFormat/specs/1.3/dataformat.1.3.html (дата обращения: 17.05.2023).

[15] KhronosGroup. KTX-Software. — 2022. — URL: https://github.com/KhronosGroup/KTX-Software (дата обращения: 17.05.2023).

[16] Kornel. How to compare images fairly. — 2014. — URL: https://kornel.ski/en/faircomparison (дата обращения: 17.05.2023).

[17] Kornel. dssim. — 2022. — URL: https://github.com/kornelski/dssim (дата обращения: 17.05.2023).

[18] LLC ImageMagick Studio. Image Similarity Comparison. — 2022. — URL: https://imagemagick.org/script/compare.php (дата обращения: 17.05.2023).

[19] McCurdy Don. glTF-Transform. — 2022. — URL: https://github.com/donmccurdy/glTF-Transform (дата обращения: 17.05.2023).

[20] Metal feature set tables. — Apple Developers, 2022. — URL: https://developer.apple.com/metal/Metal-Feature-Set-Tables.pdf (дата обращения: 17.05.2023).

[21] Nah Jae-Ho. QuickETC2: Fast ETC2 texture compression using Luma differences // ACM Transactions on Graphics. — 2020. — Vol. 39, no. 270. — P. 1–10.

[22] PVRTC Compressed Texture Image Formats. — The Khronos Group Inc., 2020. — URL: https://clck.ru/34VUZU (дата обращения: 17.05.2023).

[23] Pitaksarit Sirawat. PVRTC vs ASTC texture compression on an iOS device. — Game Torrahod, 2017. — URL: https://gametorrahod.com/pvrtc-vs-astc-texture-compression-on-an-ios-device (дата обращения: 17.05.2023).

[24] Recommended, default, and supported texture formats, by platform. — Nvidia, 2023. — URL: https://docs.unity3d.com/2023.1/Documentation/Manual/class-TextureImporterOverride.html (дата обращения: 17.05.2023).

[25] Target texture compression formats in Android App Bundles. — Android Developers, 2021. — URL: https://developer.android.com/guide/playcore/asset-delivery/texture-compression (дата обращения: 17.05.2023).

[26] Taudul Bartosz. etcpak. — 2022. — URL: https://github.com/wolfpld/etcpak (дата обращения: 17.05.2023).

[27] Texture Compression Settings. — Unreal Engine, 2023. — URL: https://docs.unrealengine.com/4.26/en-US/RenderingAndGraphics/Textures/TextureCompressionSettings/ (дата обращения: 17.05.2023).

[28] UASTC Texture Specification. — BinomialLLC, 2021. — URL: https://github.com/BinomialLLC/basis_universal/wiki/UASTC-Texture-Specification (дата обращения: 17.05.2023).

[29] Using ASTC Texture Compression for Game Assets. — Unity, 2017. — URL: https://developer.nvidia.com/astc-texture-compression-for-game-assets (дата обращения: 17.05.2023).

[30] Wang Zhou. Mean squared error: Love it or leave it? A new look at Signal Fidelity Measures // IEEE Signal Processing Magazine. — 2009. — Vol. 26. — P. 98–117.

[31] .basis File Format and ETC1S Texture Video Specification. — BinomialLLC, 2021. — URL: https://github.com/BinomialLLC/basis_universal/wiki/.basis-File-Format-and-ETC1S-Texture-Video-Specification (дата обращения: 17.05.2023).