Министерство науки и высшего образования Российской Федерации
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ (НИ ТГУ)
Институт прикладной математики и компьютерных наук

КУРСОВАЯ РАБОТА

УПРАВЛЕНИЕ ИНФОРМАЦИОННЫМИ МОДЕЛЯМИ В ВИРТУАЛЬНОЙ РЕАЛЬНОСТИ

Баканов Егор Сергеевич

Руководитель
канд. техн. наук, доцент
_____ А. В. Приступа
подпись
«____» _____ 2020 г.
Автор работы
Студент группы № 931907
_____ Е.С. Баканов
подпись

Томск – 2020

Реферат

Курсовая работа 37 стр., 26 рис., 26 источников.

Ключевые слова: информационное моделирование, виртуальная реальность, 3D визуализация, автоматизация, Unity.

Целью работы: разработка приложения визуализации и манипуляции информационными моделями в среде виртуальной реальности.

Методы проведения работ: анализ требований, проектирование системы, разработка приложения.

Полученные результаты: разработан прототип приложения, позволяющего визуализировать информационные модели, производить базовые манипуляции с трехмерным представлением модели в виртуальной реальности; частично автоматизирован процесс экспорта исходных данных информационного моделирования в формат, используемый разработанным приложением.

Содержание

Глоссарий

Меш или полигональная сетка *(англ. polygon mesh)* – структура данных, содержащая набор вершин, ребер и граней, определяющих форму многогранного объекта.

Рендеринг или отрисовка *(англ. rendering)* – процесс получения изображения двумерной или трехмерной модели с помощью компьютерной программы.

Фреймворк *(англ. framework)* – переиспользуемая, "незавершенная"система, которая может использоваться для создания другой производной системы.[1; 2]

Шейдер *(англ. shader)* – разновидность компьютерных программ, запускаемых на графических процессорах, предназначенных для отрисовки изображений.

BIM *(англ. Building Information Model)* – цифровой проект здания или другого объекта инфраструктуры, сопровождаемый базой данных всех его физических и функциональных характеристик.[3]

Unity – игровой фреймворк, используемый для трехмерной визуализации.[4]

Введение

BIM – понятие, под которым подразумевают цифровой проект здания или другого объекта инфраструктуры, которая связана с базой данных всех его физических и функциональных характеристик, содержащей подробную информацию обо всех элементах модели: элемент может содержать информацию о габаритах, поставщике и даже серийном номере. Изменения в любом элементе системы здания способны повлечь автоматические изменения параметров и объектов, вплоть до изменения чертежей, визуализаций, спецификаций, календарного графика и сметы. BIM – это общий ресурс знаний для получения информации об объекте, который служит надежной основой для принятия решений в течение всего жизненного цикла начиная с самой ранней концепции до сноса.[3] 11 июня 2016 года был утвержден список поручений Правительству Российской Федерации, направленный на развитие правовой базы использования информационного моделирования в сфере строительства.[5]

Информационное моделирование является комплексным процессом, требующим определенной компетенцией в этой области. Для использования BIM-методологии необходимы навыки использования специализированного программного обеспечения, которых может быть лишена значительная часть проектной команды. Для обычных людей крайне сложно воспринимать весь объем информации, закладываемой в BIM.

В связи с развитием технологий в последнее десятилетие произошел стремительный рост популярности технологии виртуальной реальности.[6] Как показывают многочисленные исследования, использование технологий виртуальной и дополненной реальности может улучшить производительность при валидации и верификации разрабатываемой модели. Применение технологии VR способно значительно повысить презентационные качества модели, что усилит вовлеченность в проект участников, не имеющих специальных профильных навыков.[7] Исходя из этого было принято решение о разработке приложения, способного визуализировать трехмерную репрезентацию информационной модели в VR-среде.

Цель работы – разработать прототип приложения, позволяющего инспектировать BIM модели в виртуальной реальности.

Задачи

1. реализовать извлечение атрибутивной информации модели;
2. реализовать серверную часть приложения, занимающуюся хостингом и предобработкой моделей;
3. автоматизировать перенос моделей из сред разработки в приложение.
4. реализовать модуль взаимодействия пользователя с моделью на клиентской части приложения;

Стоит отметить, что данный проект разрабатывается командой из нескольких человек, поэтому в ходе работы не будут представлены те части, в которых автор не принимал непосредственного участия при разработке.

На момент написания этого отчета разрабатываемый прототип не получил официального названия, которое могло бы использоваться при разработке или коммерческом продвижении. Поэтому вместо этого будет использоваться его неофициальное внутреннее название – **BIMExplorer**.

Глава 1. Аналитика

Данный раздел содержит обзор существующих решений, направленных на визуализацию информационных моделей в виртуальной реальности. В ходе их анализа были выявлены функциональные и нефункциональные требования к реализации системы.

1.1 Существующие решения

В ходе изучения существующих решений был обнаружен ряд продуктов как в индустриальной, так и в академической среде. Ниже приведены несколько примеров, на основе которых были сформулированы требования к разрабатываемому прототипу.

Индустриальная среда

Unity Reflect – приложение разрабатываемое компанией Unity Technologies, на основе их игрового фреймворка Unity.[8] Reflect обладает интеграцией с несколькими программами информационного моделирования, такими как Revit, Rhino и Sketchup. Reflect способен синхронизировать изменения информационной модели с ее VR отображением в реальном времени, а также извлекать атрибутивную информацию. К сожалению на момент разработки нашего решения и написания этой статьи Unity Reflect был не завершенным продуктом, находящимся в активной разработке.

Prospect – аналогичное приложение от компании IrisVR, поддерживающее информационные модели из Navisworks, Revit, Rhino, Sketchup, а также 3D форматы FBX и OBJ.[9] Prospect способен извлекать атрибутивную информацию, имеет достаточно высокую производительность, а также возможность проведения многопользовательских сессий.

Enscape – ещё одно приложение с интеграцией с Revit, Sketchup, Rhino, ArchiCAD и Vectorworks.[10] Enscape может синхронизировать изменения информационной модели в реальном времени и извлекать атрибутивную информацию. Из особенностей Enscape стоит отметить широкий набор настроек и эффектов отображения информационной модели.

Академическая среда

В статье Джордана Дэвидсона и др. описана разработка прототипа приложения на основе Enscape расширения для Revit, описанного раннее.[11] Целью работы было расширение уже имеющихся возможностей Enscape для экспериментальной проверки пользовательского опыта при инспектировании модели. Приложение Дэвидсона отличается от предыдущих тем, что обладает обратной связью с отображаемой информационной моделью, позволяющей пользователю прямо внутри VR симуляции изменять окружение. В рамках прототипа эта особенность ограничивалась изменением внутреннего интерьера, мебели и характеристик окон. Как утверждает автор, данный подход

повышает вовлеченность клиента в проект на ранних стадиях разработки, повышает его осведомленность о решениях, принятых архитекторами, и снижает риск изменений в “последнюю минуту”, повышающих стоимость реализации проекта.

Ещё одно возможное решение задачи описал в своей работе Фарзад Пур Рахимян и др.[12] Помимо частичной обратной связи, аналогичной той, что представлена в работе Джордана Дэвидсона, прототип Рахимяна уникален использованием в качестве целевого формата информационных моделей Industry Foundation Classes – открытый, международный и независимый от других производителей стандарт, разработанный buildingSMART.[13] В работе также описана клиент-серверная архитектура решения: на клиентской части приложения происходит интерактивная демонстрация модели в виртуальной или дополненной реальности; сервер же является промежуточным слоем между исходной информационной моделью и визуализацией, отслеживающим изменения и производящим синхронизацию данных.

1.2 Требования к системе

Далее приведен формализованный список требований к разрабатываемому прототипу.

Нефункциональные требования

- совместимость с форматом Revit, как наиболее распространенной системы информационного моделирования;
- разделение функциональности приложения на “серверную” и “клиентскую” для повышения гибкости системы к возможным дальнейшим расширениям и независимости пользователя от конкретных систем информационного моделирования;
- высокая производительность при инспектировании комплексных моделей;
- извлечение атрибутивной информации, закладываемой в модель.

Стоит отметить, что в рамках прототипа не производилась реализация полноценного удаленного сервера, а производилась эмуляция его работы локально.

Функциональные требования

- выбор BIM модели из уже загруженных на сервер;
- изменение масштаба отображения модели между натуральной величиной и размерами настольного макета;
- управление отображением различных слоев модели, созданных на основе извлеченной атрибутивной информации;

Особенности реализации пользовательского интерфейса и функциональности навигации в виртуальном пространстве будут опущены, так как реализовывались другими членами команды разработки прототипа.

Глава 2. Проектирование

Этот раздел содержит описание теоретических аспектов работы. В первую очередь здесь будет детально рассмотрена структура информационных моделей, используемых как источник исходных данных для визуализации. Далее идет описание клиентской и серверной части прототипа, описание проблем, которых это разделение призвано решить, а также описание функциональности, которую они будут предоставлять. Наконец, эта глава также содержит кратких обзор методик, используемых для повышения производительности визуализации.

2.1 Описание предметной области

Далее рассматривается структура информационной модели. В связи с тем, что в качестве целевого формата был выбран формат Revit, дальнейшее описание предметной области будет в первую очередь соответствовать стандартам Autodesk Revit. Различия с другими форматами, например Industry Foundation Classes (IFC-формат),[13] рассматриваться не будут.

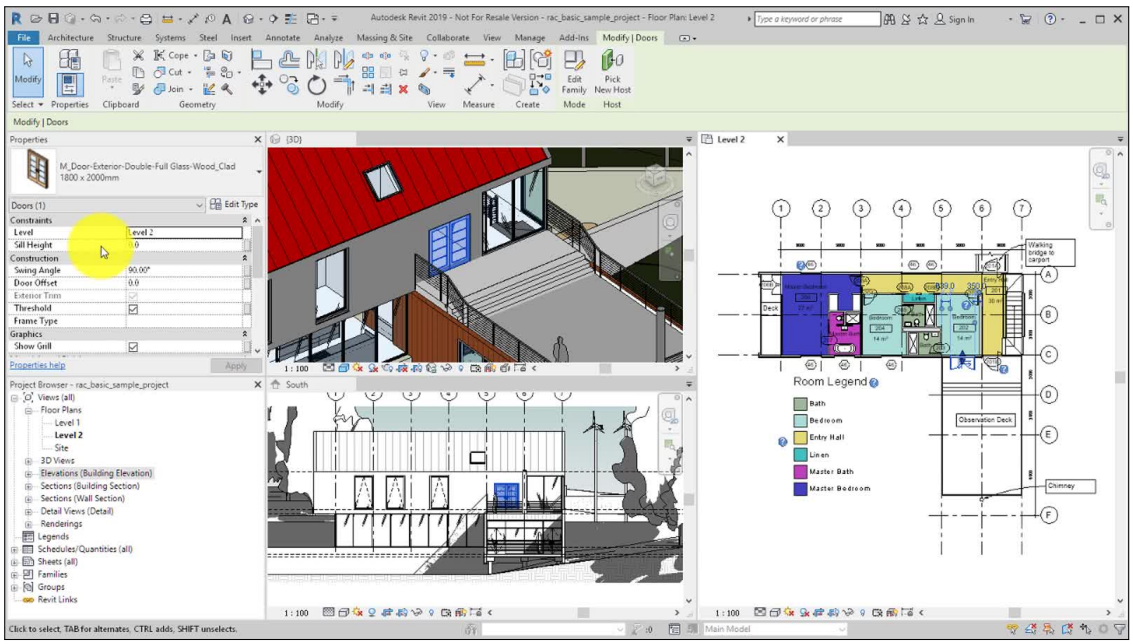

Рис. 1: Интерфейс редактора Revit.[14]

Для хранения данных Revit использует специальный проприетарные форматы RVT, RFA и RTE. Для отображения данных модели в виртуальной среде необходимо преобразовывать их в форматы трехмерной графики, например OBJ или FBX.

Элементы информационной модели

Каждая сущность в проекте информационной модели называется элементом. Revit использует 3 типа элементов в проектах: элементы модели, опорные элементы и элементы, относящиеся к представлению. Элементы в Revit также часто называются

семействами. Семейство содержит геометрическое определение элемента и параметры, используемые элементом. Каждый экземпляр элемента определяется и контролируется семейством.[14] Для создания визуализации нас в первую очередь интересуют именно элементы модели.

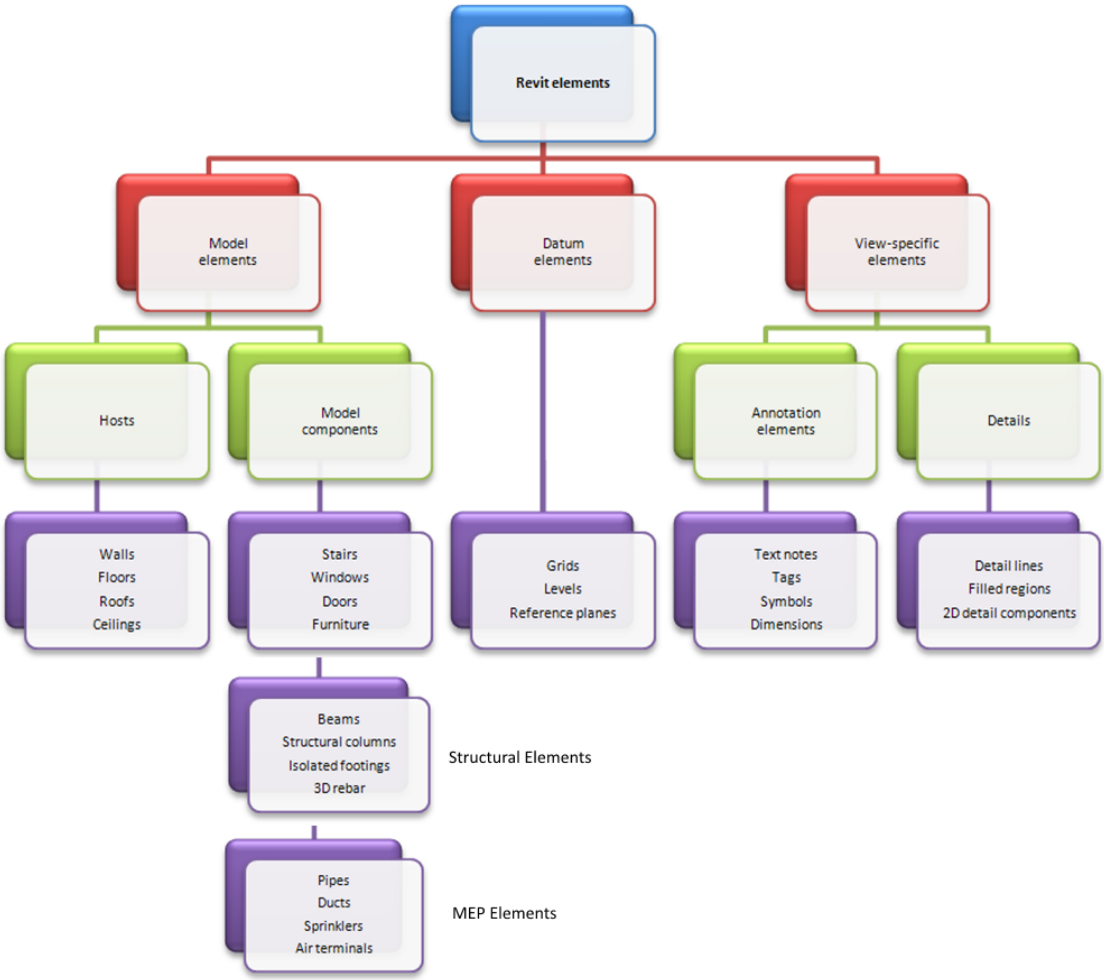

Рис. 2: Элементы информационной модели.[14]

- Элементы модели.

  Элементы модели представляют фактическую трехмерную геометрию здания, например стены, окна, двери, пандусы, воздуховоды и электрические панели. Элементы модели делятся на хосты и компоненты модели. Хостами обычно являются элементы, которые возводятся непосредственно на строительной площадке, например стены и потолки. Компонентами модели являются все остальные типы элементов модели.

- Опорные элементы.

  Опорные элементы помогают определить контекст проекта. К ним относятся опорные плоскости, уровни и сетки.

- Элементы представления.

  Элементы представления это элементы, отображаемые только в каком-то определенном режиме представления проекта. Они помогают описывать или документировать модель.

Свойства элементов

Каждый размещаемый элемент является экземпляром семейства. Элементы имеют 2 набора свойств, управляющих их внешним видом и поведением: свойства типа и свойства экземпляра.[14]

- Свойства типа.

  Один и тот же набор свойств типа является общим для всех элементов семейства, и каждое свойство имеет одинаковое значение для всех экземпляров определенного типа семейства. Изменение значения свойства типа влияет на все текущие и будущие экземпляры этого семейства.

- Свойства экземпляра.

  Общий набор свойств экземпляра также применяется ко всем элементам, принадлежащим к определенному типу семейства, но значения этих свойств могут различаться. Например, размеры окна являются свойствами типа, а его высота расположения над уровнем пола является свойством экземпляра.

2.2 Описание структуры клиентской и серверной части

Теперь можно поподробнее рассмотреть архитектуру прототипа. Как было сказано ранее, основное предназначение приложения – презентация информационной модели участникам проекта, не имеющим навыков моделирования, например заказчикам, спонсорам или непосредственным пользователям здания. Такая презентация поможет успешнее вносить корректировки в проект еще на самых ранних стадиях, что снизит финальные затраты на реализацию проекта.[11]

Исходя из описанных требований можно сделать вывод о том, что конечные пользователи приложения не станут использовать программное обеспечение информационного моделирования. Для них имеют значение только финальные или промежуточные результаты работы. Поэтому было принято решение отделить функциональность приложения, взаимодействующую с исходными данными модели. Клиентская часть может не зависеть от конкретной системы информационного моделирования, что упростит расширение системы в дальнейшем. Так же это может повысить производительность визуализации в клиентской части, так как все трудоемкие вычисления, проводимые над исходными данными, можно проводить на серверной части.

Функциональность серверной части

Далее представлена основная функциональность реализуемая прототипом на "сервере".

- Хранение исходной информационной модели.

  Сервер должен хранить исходную модель, а также распознавать ее изменения для повторного преобразования.

- Конвертация формата RVT в формат FBX.

  Поскольку формат данных информационной модели отличается от представления классических форматов трехмерной графики и не поддерживается большинством фреймворков создания интерактивных графических приложений вроде Unity и Unreal, необходимо производить преобразование формата.

- Предоптимизация модели.

  Из-за комплексности трехмерных моделей, получаемых после конвертации, необходимо проводить их дополнительную оптимизацию. Подробнее это будет рассмотрено в разделе 2.3.

- Упаковка моделей.

  Оптимизированные модели должны упаковываться для загрузки в клиентской части приложения.

Функциональность клиентской части

Далее описана функциональность клиентской части приложения вместе с необходимыми схемами. Функциональность, реализуемая не автором отчета, а другими членами команды разработки, была опущена.

- Загрузка готовой модели.

  Приложение должно загружать выбранную упакованную модель. Так как в рамках прототипа не производится реализация удаленного сервера, загрузка будет происходить локально, как показано на рисунке 3.

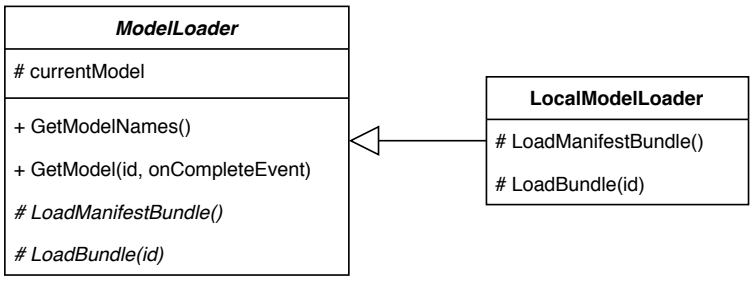

Рис. 3: Загрузчик моделей.

Как видно из схемы, для загрузки используется абстрактный класс ModelLoader, обладающий двумя публичными методами: метод, позволяющий получить список доступных для загрузки информационных моделей, и метод, асинхронно загружающий выбранную информационную модель по ее индикатору. Для непосредственной загрузки списка доступных моделей и самих моделей используются два внутренних метода, реализуемые в подклассах ModelLoader'a. Например в рамках прототипа будет реализован класс LocalModelLoader, загружающий информационные модели с локального дискового устройства, что подробнее рассмотрено в разделе 3.3.

- Размещение модели в виртуальной среде.

  После загрузки модель должна размещаться в виртуальной сцене. Для визуализации будет использоваться виртуальный стол, на который будет проецироваться модель здания в уменьшенном масштабе согласно схеме на рисунке 4.

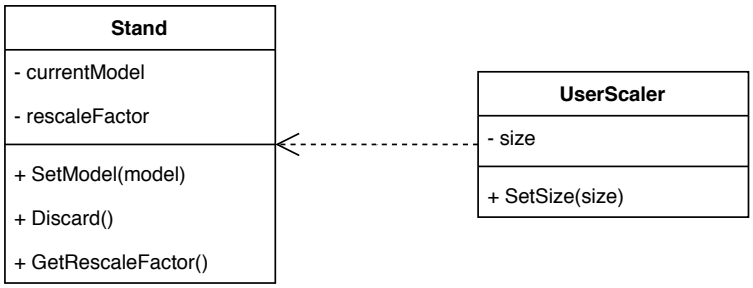

Рис. 4: Размещение модели.

Для корректного размещения модели используется класс Stand, отвечающий за расчет размеров модели. Размещаемая модель должна полностью использовать поверхность виртуального стола, не выходя за его пределы. После размещения модели пользователь приложения имеет возможность изменять свой размер в виртуальном пространстве, чтобы соответствовать размерам модели. Для этого используется класс UserScaler, позволяющий изменять размер пользователя через передачу числового параметра от 0 до 1, где 0 соответствует масштабу информационной модели, а 1 – изначальному размеру пользователя в виртуальной среде. Подробности реализации описаны в разделе 3.3.

- Взаимодействие с элементами модели.

  После размещения модель будет разбивать на слои, представляющие отдельные структурные элементы здания, например стены, лестницы или двери. На каждый отдельный слой может накладываться эффект, изменяющий его отображение. На рисунке 5 показана структура системы слоев.

  Класс Layer представляет слой информационной модели. Он обеспечивает доступ ко всем необходимым для визуализации данным, позволяя модифицировать их

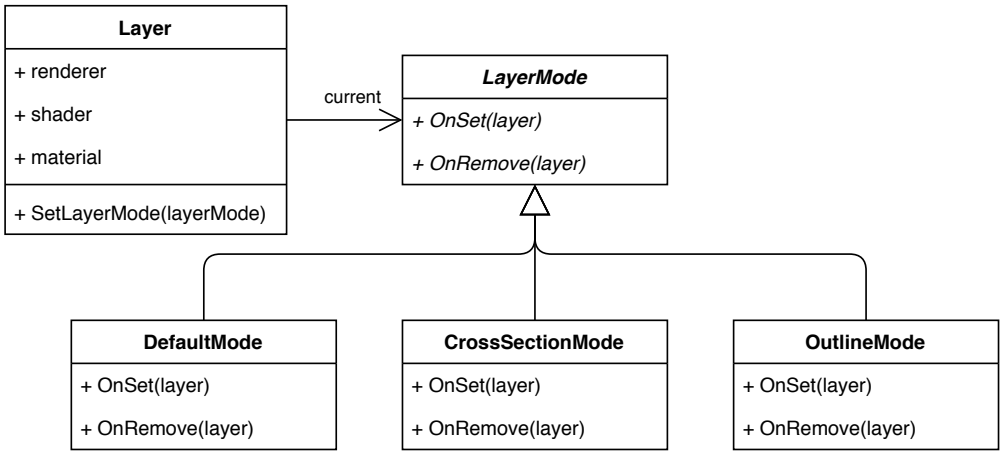

Рис. 5: Система слоев.

при изменении режима отображения слоя (абстрактный класс LayerMode). Подробности реализации различных режимов отображения приведены в разделе 3.3.

2.3 Обзор оптимизационных подходов

Теперь можно рассмотреть подходы повышения производительности визуализации. Важность этого аспекта была выявлена ещё на самых ранних стадиях разработки прототипа. Информационные модели зачастую обладают очень комплексной геометрией. Более того, визуализация в виртуальной реальности накладывает дополнительные ограничения, так как изображение проходит двукратную отрисовку из-за наличия двух глаз с разной перспективой. В добавок к этому низкая частота кадров в виртуальной реальности может вызывать плохое самочувствие у пользователя.[15]

Вспомогательные понятия

Для начала введем несколько вспомогательных терминов:

Графический материал – набор данных, ассоциируемый с моделью и определяющий то, как будет отрисована ее поверхность в зависимости от выбранного шейдера.

Меш или полигональная сетка *(англ. polygon mesh)* – структура данных, содержащая набор вершин, ребер и граней, определяющих форму многогранного объекта.

Шейдер *(англ. shader)* – разновидность компьютерных программ, запускаемых на графических процессорах, предназначенных для отрисовки изображений.

Draw call *(рус. Вызов отрисовки)* – команда отрисовки полигональной сетки, отдаваемая центральным процессором графическому.

Распространенные причины проблем с производительностью

- Сложная геометрия модели. Под сложной геометрией понимается большое количество полигонов (граней) в полигональной сетке модели.
- Большое количество draw call'ов. Выполнение запроса на отрисовку выполняется центральным процессором и является достаточно трудоемкой операцией. За один вызов может быть обработана только одна полигональная сетка. Вполне возможна ситуация, при которой центральный процессор тратит больше времени, чтобы инициировать отрисовку, чем графический процессор будет ее исполнять.
- Трудоемкие шейдеры. Алгоритм шейдера может иметь высокую вычислительную сложность (например использовать долговычислимые тригонометрические функции) или иметь неподходящую структуру для запуска на графическом процессоре (например содержать многочисленные ветвления). Помимо этого работа шейдера может предполагать неоднократную отрисовку объекта для достижения определенного результата.

Существующие подходы

- Visibility culling (Отбор видимых полигонов).

  Visibility culling – это семейство алгоритмов, нацеленное на предотвращение вызовов отрисовки для объектов невидимых в кадре.[16] Пример различных техник отбора видимых полигонов показан на рисунке 6. Подобные алгоритмы являются очень эффективными, когда количество объектов единовременно видимых в виртуальной сцене значительно меньше их общего количества, например в замкнутых помещениях внутри крупного здания.

  - View-frustum culling – это техника отбора, отсекающая отрисовку объектов, находящихся за границами поля зрения виртуальной камеры.
  - Back-face culling – это техника отбора, позволяющая избежать отрисовку геометрии, направленной в противоположную сторону от виртуальной камеры.
  - Occlusion culling – это техника отбора, предотвращающая отрисовку геометрии, скрытой за другими объектами виртуальной сцены.

- LOD (англ. Level of detail – уровень детализации).

  LOD – это оптимизационная техника, нацеленная на снижение используемого количества полигонов модели при ее удалении от виртуальной камеры. Для работы метода требуется создать несколько версий одной и той же модели с разным уровнем детализации (количеством полигонов), как это показано на рисунке 7.

  Перед вызовом запроса отрисовки производится выбор необходимой модели, то есть вблизи от виртуальной камеры будут отрисовываться детализированные модели (LOD0), а вдали упрощенные (LOD1-LOD3).

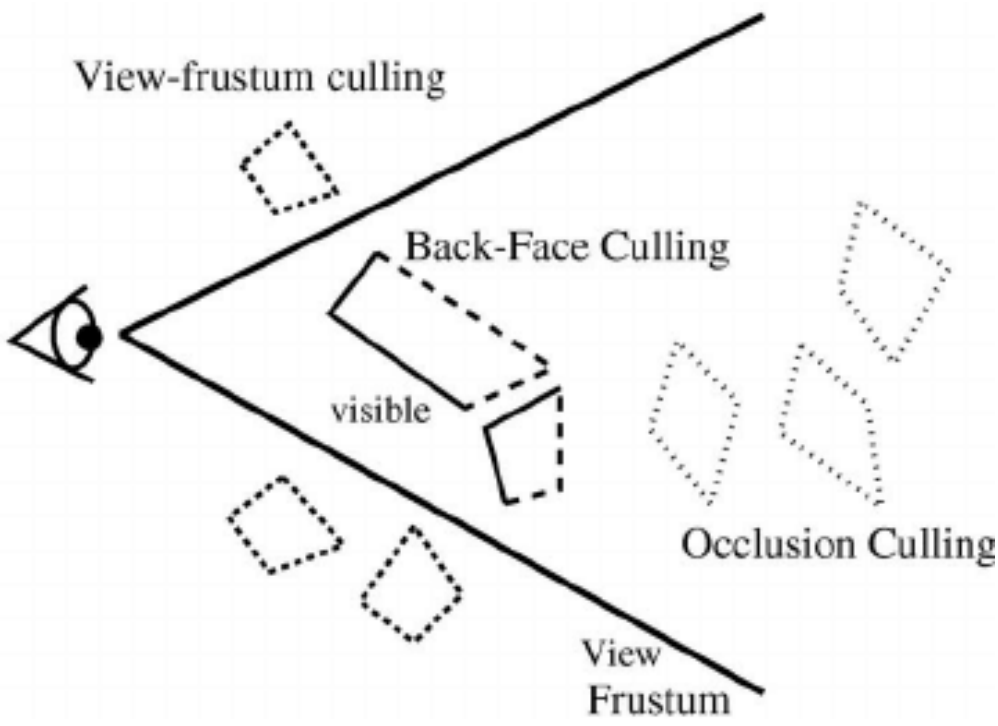

Рис. 6: Техники отбора видимых полигонов.[16]

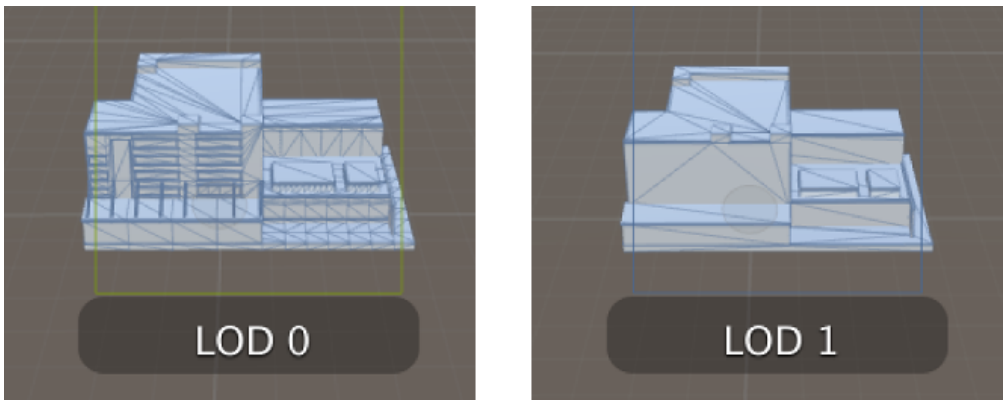

Рис. 7: Разные уровни детализации.[4]

- Batching.

  Батчинг – это оптимизационная техника, предназначенная для снижения количества запросов отрисовки за счет группировки нескольких полигональных сеток.

  - Батчинг может происходить динамически, если группируется множество простых объектов, с одинаковыми графическими материалами и текстурами (чего можно добиться с помощью создания атласа текстур). За счет этого можно отрисовывать несколько движущихся объектов за один вызов отрисовки, что имеет смысл, если продолжительность группировки меньше, чем затраты на многочисленные вызовы отрисовки.
  - С другой стороны батчинг может быть статическим, то есть происходить однократно и объединять неизменяемую геометрию в одну полигональную сетку, позволяя достичь ещё большей производительности, чем при динамической группировке. Стоит отметить, что подобная группировка снижается эффективность отбора видимых полигонов, так как сгруппированные объекты начнут восприниматься как один.

- Geometry instancing (Дублирование геометрии).

  Geometry instancing – это методика, позволяющая одновременно рендерить несколько объектов, имеющих одинаковую полигональную сетку. Такой подход в основном используется для отрисовки повторяющихся фоновых объектов, таких как растительность или здания. Объекты могут иметь разное положение в пространстве, размеры, отличающиеся графические материалы. Дублирование геометрии значительно снижает количество запросов на отрисовку.

Выбранный подход

При разработке прототипа было принято решение использовать вариацию статического батчинга, при котором полигональные сетки всех объектов одного слоя (описано ранее в разделах 2.1 и 2.2) объединяются в одну на одном из этапов конвертации исходной информационной модели. В дальнейшем этот подход может быть пересмотрен, так как несмотря на продемонстрированную эффективность он обладает рядом недостатков, а именно пропадает возможность взаимодействия с отдельными объектами слоя, а также теряется значительная часть атрибутивной информации.

Глава 3. Реализация

Данная глава содержит детали технической реализации прототипа. В первую очередь будет приведен перечень инструментов, используемых при реализации. Затем идет описание реализации серверной и клиентской части прототипа.

3.1 Обзор инструментов

Unity

В качестве фреймворка для выполнения проекта был выбран Unity. Unity – это фреймворк, предназначенный для разработки интерактивных графических приложений.[4] Первая версия фреймворка была выпущена в 2005 году и с тех пор продолжает активно развиваться. Для создания приложений Unity поддерживает более 20 платформ, включающих персональные компьютеры, мобильные устройства, игровые консоли и др. Фреймворк может быть использован для создания приложений с двумерной или трехмерной графикой, а также приложений в виртуальной или дополненной реальности.

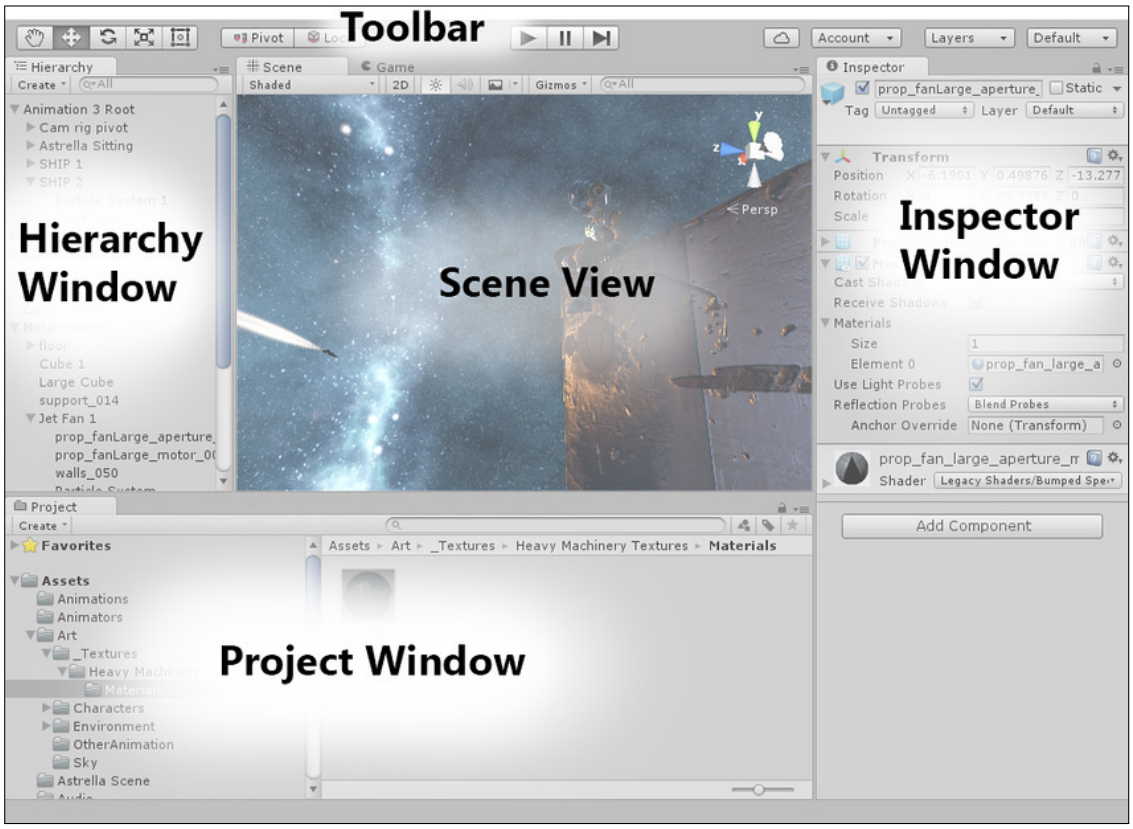

Рис. 8: Графический интерфейс редактора Unity.[4]

Unity имеет встроенные модули для работы с графикой, пользовательским вводом, физикой, пользовательским интерфейсом и сетевым взаимодействием. Компонентная архитектура фреймворка позволяет легко расширять уже существующую функциональность.

Основное направление использования Unity – это разработка игр, тем не менее Unity также успешно применяется в киноиндустрии,[17] архитектуре, автомобилестроении,[18] разработке виртуальных тренажеров, а также в обучении искусственного интеллекта.[19]

C#

C# – высокоуровневый мультипарадигменный язык общего назначения, разработанный компанией Microsoft в 1998-2001 годах в рамках платформы .NET Framework. C# относится к семье языков с C-подобным синтаксисом и используется для написания веб-приложений, приложений для персональных компьютеров и мобильных устройств. [20] C# является основным скриптовым языком фреймворка Unity, тем самым он используется для написания всей клиентской и серверной логики разрабатываемого прототипа.

SteamVR

SteamVR – это инструмент, унифицирующий работу с различными устройствами виртуальной реальности, разработанный компанией Valve Corporation.[21] SteamVR обладает плагинами для фреймворков Unity и Unreal Engine, что позволяет интегрировать его в разрабатываемый прототип.

HTC Vive

HTC Vive – это шлем виртуальной реальности, разрабатываемый компаниями HTC и Valve Corporation.[22] HTC Vive используется во время тестирования прототипа.

3ds Max

3ds Max – профессиональное программное обеспечение для 3D-моделирования, разрабатываемое компанией Autodesk. 3ds Max является расширяемым инструментом, что позволяет частично автоматизировать часть процессов через создание дополнительных плагинов.[23] 3ds Max используется в процессе преобразования информационной модели из rvt-формата в формат fbx.

3.2 Серверная часть

Данный подраздел содержит описание реализации серверной части приложения. В рамках прототипа было принято решение разработать скрипт, который, подобно конвейеру, обрабатывает исходную информационную модель для производства пакета, содержащего все необходимые данные (трехмерное представление здания, пригодное для производительного рендеринга; графические материалы; извлеченная атрибутивная информация), который может загружать используемый фреймворк визуализации.

Конвертация информационной модели в FBX-формат

На первой стадии обработки производится преобразование формата модели. Как было сказано в разделе 2.1, исходный RVT-формат является проприетарной закрытой разработкой компании Autodesk Inc., хранящей данные в бинарном формате, что сильно затрудняет разработку парсеров для этого формата. Помимо этого, Autodesk использует свой формат графических материалов, которые также плохо-поддерживаются современными графическими фреймворками.

Наиболее эффективными инструментами по работе с форматами компании Autodesk Inc. являются её же разработки, поэтому для выполнения преобразования был выбран редактор 3ds Max. 3ds Max обладает всей требуемой функциональностью, но, к сожалению, его программный интерфейс (API) не обладает полным покрытием этой функциональности, поэтому этот этап не удалось полностью автоматизировать при разработке прототипа.

Использование 3ds Max для конвертации предполагает следующие шаги:

1. Загрузка исходной информационной модели.

   Для загрузки используется инструмент File Link Manager (рисунок 9), позволяющий загружать в 3ds Max файлы форматов DWG, DXF, FBX и RVT. Для загрузки

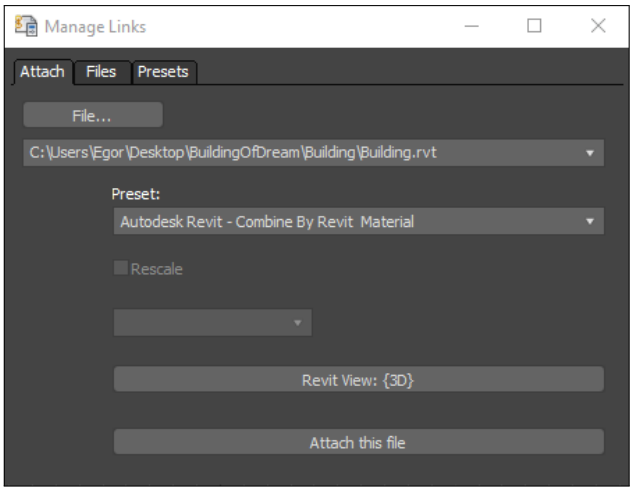

Рис. 9: File Link Manager

   необходимо выбрать загружаемый файл (Attach/File...), а затем выбрать одну из предустановленных опций загрузки (Preset). Всего на данный момент существует 5 опций, но в рамках прототипа нас интересуют только 2:

   - Autodesk Revit - Do Not Combine Entities

     При импорте с этой опцией отдельные элементы информационной модели будут иметь отдельные полигональные сетки, что в дальнейшем позволит полностью сохранить всю атрибутивную информацию. Недостатком этой опции является низкая финальная производительность и большее использование памяти для хранения такой модели на диске и в оперативной памяти.

- Autodesk Revit - Combine By Revit Category

  При импорте с данной опцией элементы модели, относящиеся к одной категории, будут объединены в одну полигональную сетку, что можно считать статическим батчингом модели (описано ранее в разделе 2.3). Такой вариант был выбран предпочтительным, хотя при нём неизбежно утрачивается значительная часть атрибутивной информации.

После этого необходимо завершить загрузку: Attach/Attach this file + Files/Bind...

2. Конвертация графических материалов.

   Для конвертации графических материалов сцены был использован инструмент Scene Converter (рисунок 10). Scene Converter позволяет конвертировать различ-

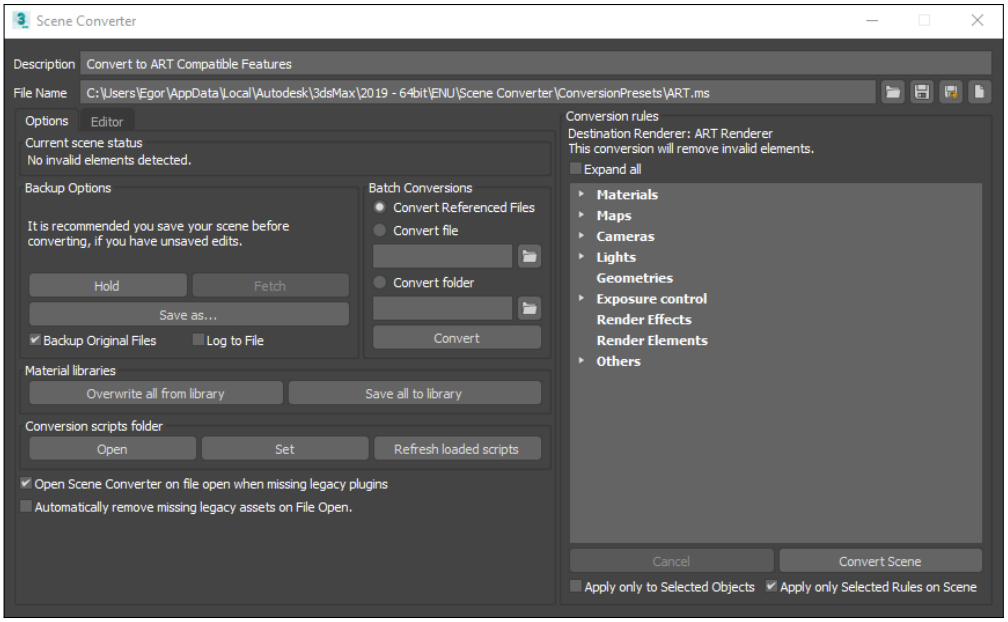

Рис. 10: Scene Converter

   ные элементы сцены, применяя определенные правила. В данном случае нас интересует правило “Autodesk Material to Physical Material”, преобразующее проприетарные графические материалы Autodesk, в широко используемые графические материалы, основанные на физических свойствах поверхности.

3. Сохранение FBX модели.

   В завершение необходимо экспортировать полученную модель в FBX формате (рисунок 11). При сохранении необходимо исключить все анимации, камеры и источники света, если они присутствуют в оригинальной информационной модели, но сохранить встроенные медиа-данные.

Загрузка модели в промежуточный проект

Приложения, разработанные на основе фреймворка Unity не имею встроенной возможности к загрузке произвольных трехмерных моделей непосредственно с диско-

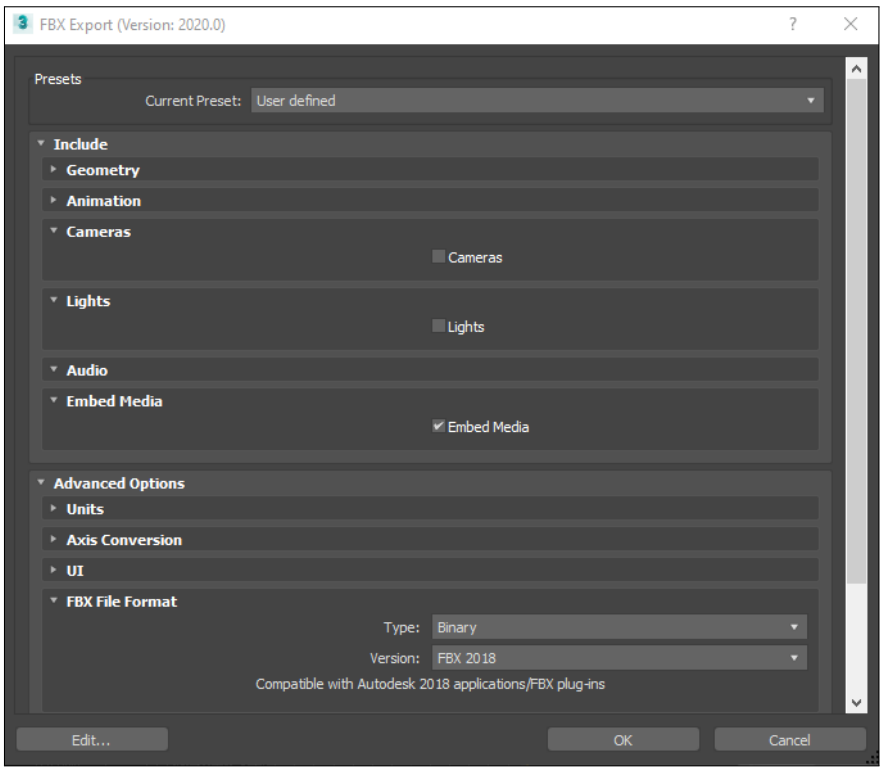

Рис. 11: FBX Export

вого накопителя. Модели в Unity могут загружаться одним из следующих способов:

- Модель должна быть непосредственно загружена в проект разрабатываемого приложения до его компиляции. После этого она может загружаться в запущенном приложении за счет внутреннего механизма сериализации данных. Данный подход очевидно является неприменимым в нашем случае, так как он приведет к встраиванию информационной модели в приложение.

- Загрузка модели может осуществляться за счет специализированного отдельно разработанного модуля, включенного в клиентскую часть прототипа. Такое решение будет слишком трудоемко для разработки в рамках прототипа.

- Загрузка моделей может производиться с помощью встроенной в фреймворк системы динамической загрузки стороннего контента. Загружаемый контент должен предварительно упаковывать в специальные файлы, создание которых возможно только с помощью редактора Unity.

Таким образом, вторым этапом обработки модели является подготовка модели к упаковке через ее добавление в промежуточный проект редактора Unity. К счастью, программный интерфейс редактора Unity полностью покрывает все наши нужды, тем самым весь дальнейший процесс поддается полной автоматизации.

Для автоматического импортирования моделей в промежуточный проект необходимо создать скрипт, который можно запускать из командной строки без участия пользователя.[4] Общий алгоритм импорта моделей показан на рисунке 12.

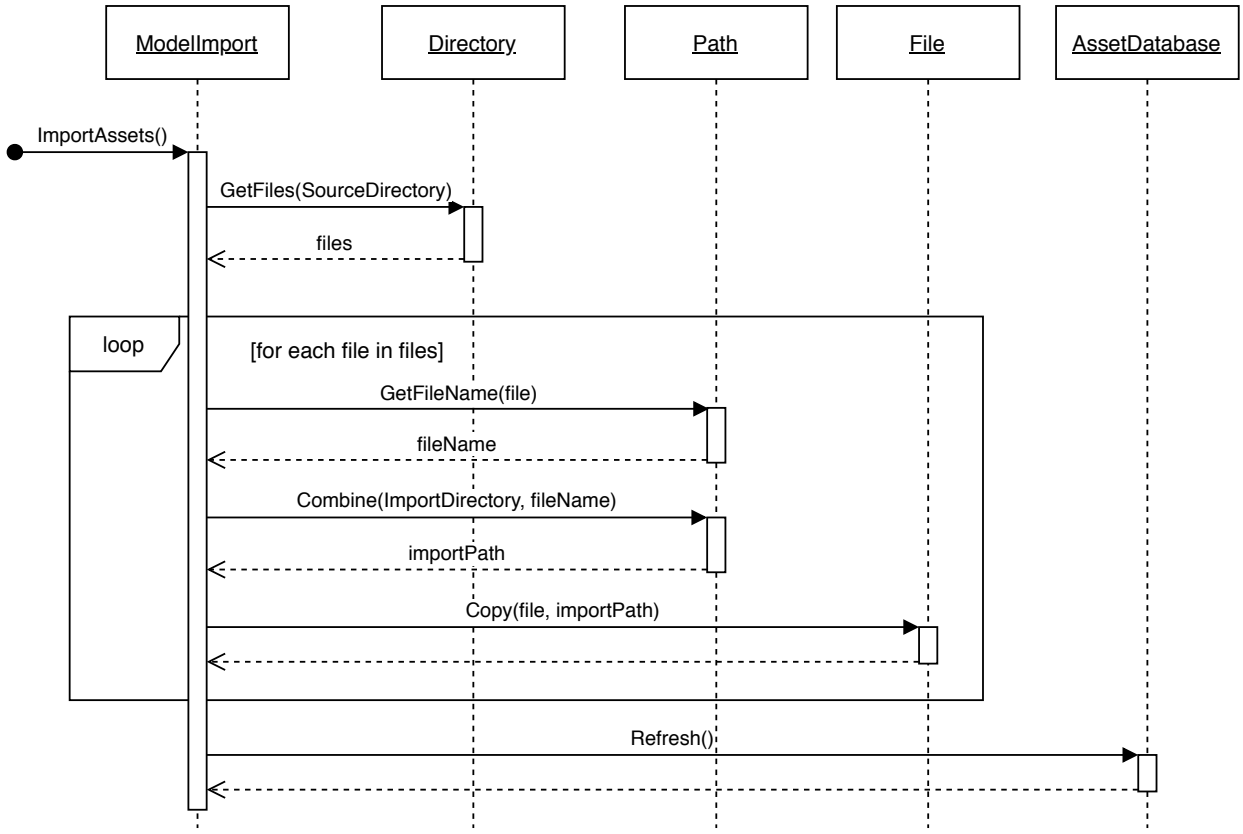

Рис. 12: Импорт моделей.

Для импорта новых ресурсов в проект Unity необходимо скопировать файлы в директорию проекта (*<путь к проекту>/Assets*) и вызвать метод обновления базы данных ресурсов Unity (*AssetDatabase*), что приведет к импорту всех новых или измененных ресурсов, а также отчистке базы данных от уже несуществующих.[4]

На стадии импорта моделей можно также проводить извлечение сохранившейся в модели атрибутивной информации. Для этого нам потребуется создать специальный обработчик моделей (рисунок 13).

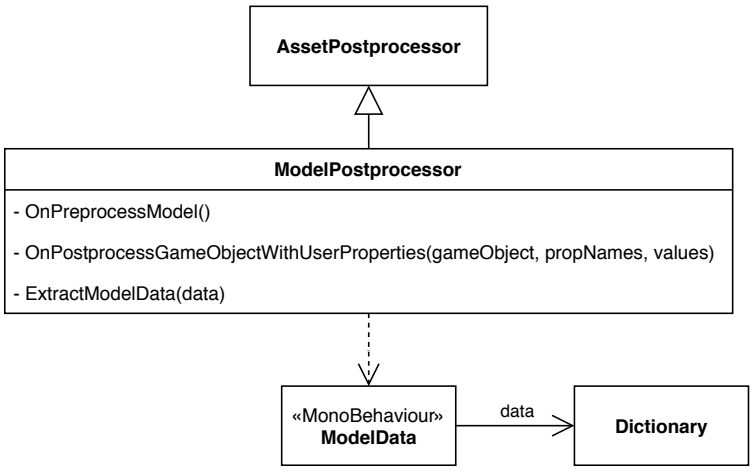

Рис. 13: Постобработчик моделей.

Для обработки ресурсов на различных стадиях импорта в проект Unity, например

до начала импорта или после его завершения при определенных условиях, в проекте реализуется наследник класса *AssetPostprocessor*. Для задания необходимого поведения при обработке ресурсов в классе-наследнике реализуются набор специальных методов: *OnPreprocessModel* – метод, позволяющий задать настройки импорта до его начала (в нашем случае необходимо задать генерацию данных для внутренней системы Unity по обработке физических коллизий); *OnPostprocessGameObjectWithUserProperties* – метод, позволяющий обработать импортированный ресурс, если у него были найдены "определенные пользователем свойства", что в нашем случае является атрибутивной информацией модели.[4] Извлеченная атрибутивная информация сохраняется в вспомогательно объекте (*ModelData*) в формате ключ-значение. Сам процесс извлечения атрибутивной информации показан на рисунке 14.

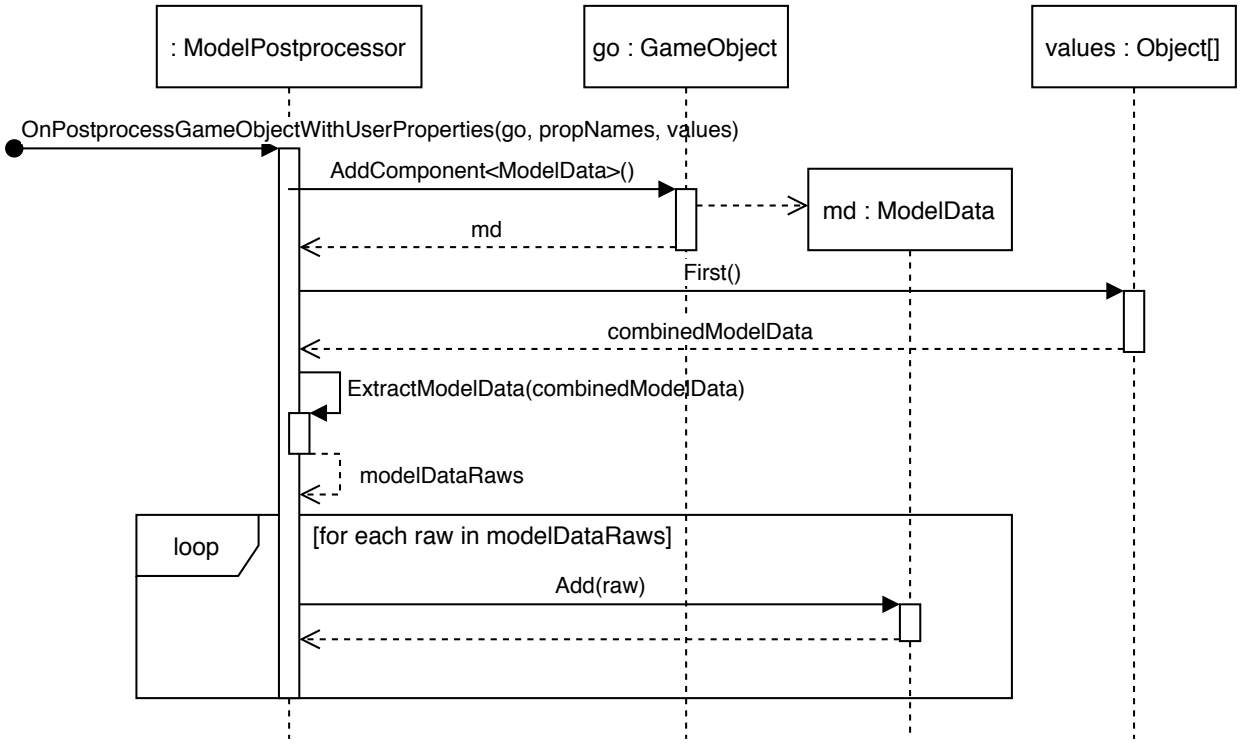

Рис. 14: Извлечение атрибутивной информации модели.

Извлекаемая атрибутивная хранится в первом элементе входного массива *values* в текстовом виде.[4] Для извлечения информации из полученной строки используется метод *ExtractModelData*, возвращающий извлеченную атрибутивную информацию в виде массива ключ-значение. После чего извлеченная информация сохраняется в компоненте *ModelData*, добавленном к обрабатываемому объекту.

Упаковка моделей

Финальным шагом обработки информационных моделей является упаковка. Для этого был выбран механизм фреймворка Unity называющийся AssetBundle. AssetBundle это архив, содержащий не скриптовые ресурсы, например модели или текстуры, который можно динамически загружать или выгружать в момент выполнения приложения.

Пакеты могут обладать взаимными зависимостями, например графический материал, упакованный в один AssetBundle может использовать текстуру, упакованную в другой AssetBundle (рисунок 15), что может сильно уменьшить расход памяти, если таких пакетов с графическими материалами станет несколько.[4; 24]

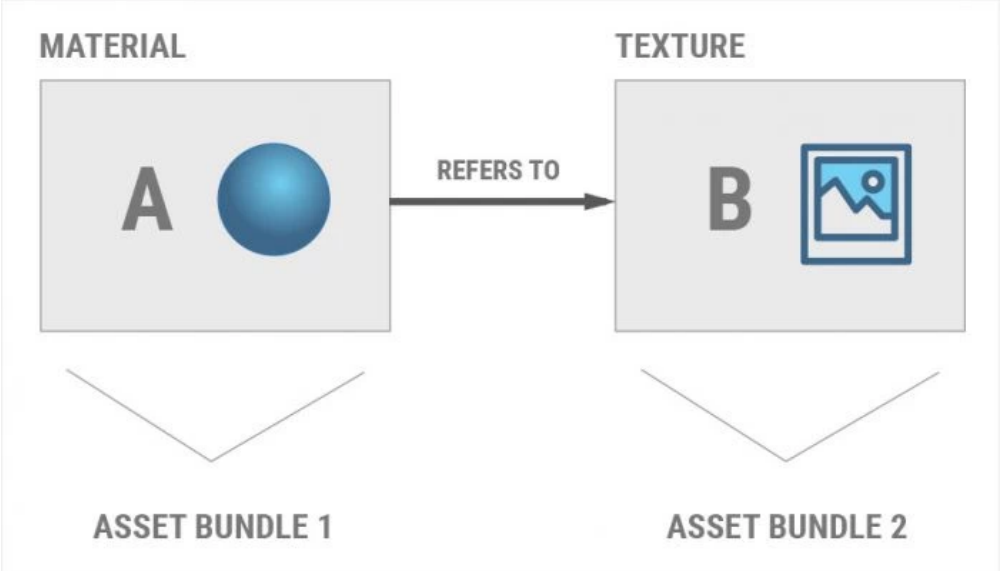

Рис. 15: Взаимозависимости ресурсных пакетов.[24]

Помимо этого AssetBundle обладает встроенными механизмами сжатия, а также механизмами версионности и кеширования, что позволит эффективнее справляться с модификациями информационной модели.[4; 24] Создание ресурсных пакетов тоже может быть полностью автоматическим, что показано на рисунке 16.

Для создания AssetBundle'ов необходимо вызвать соответствующий метод у класса, отвечающего за программное инициирование сборки проекта Unity (*BuildPipeline*). В методе необходимо указать конечную директорию, в которой будут размещены пакеты, а также информацию по каждому создаваемому пакету, включающую название пакета, перечень ресурсов, включаемых в пакет, а также соответствующие ресурсам имена, используемые при извлечении ресурсов во время работы приложения.[4; 24]

3.3 Клиентская часть

Данный подраздел содержит описание реализации клиентской части приложения, а также демонстрацию его работы. При разработке прототипа была реализован загрузка и размещение модели, а также функционал нескольких режимов отображения слоев модели, как это было описано в разделе 2.2.

Загрузка модели

Первой функцией клиентской части является загрузка информационной модели, прошедшей процесс упаковки (раздел 3.2). Как говорилось ранее в разделе 2.2, в

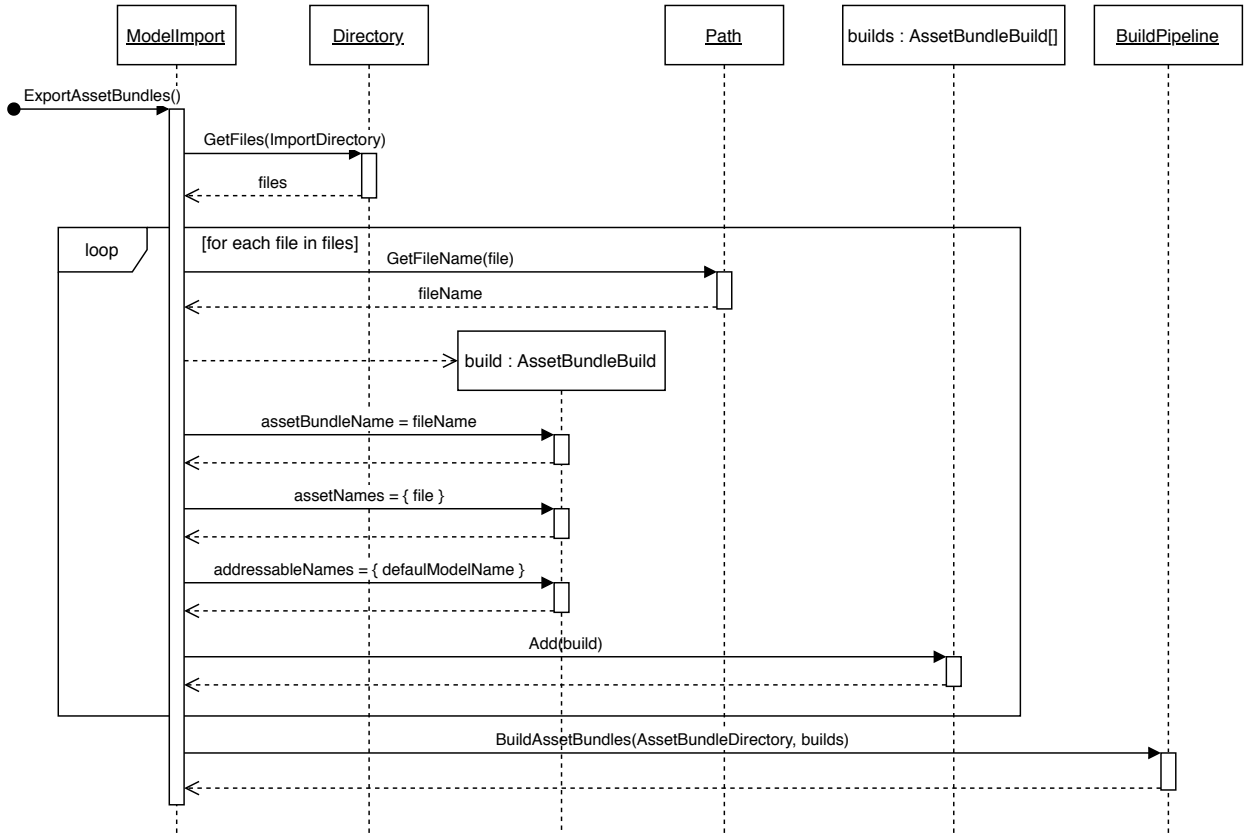

Рис. 16: Экспорт пакетов

рамках прототипа реализована загрузка упакованной модели с локального дискового устройства.

Для определения того, какие информационные модели прошли упаковку и доступны для загрузки, необходимо сначала загрузить дополнительный AssetBundle, автоматически создаваемый в процессе упаковки моделей и называемый в фреймворке Unity манифестом ресурсных пакетов. Этот AssetBundle содержит перечень всех пакетов, созданных во время экспорта.[4; 24] Загрузка манифеста будет осуществляться при инициализации загрузчика, как это показано на рисунке 17.

Механизм инициализации загрузчика представляет собой асинхронную загрузку ресурсного пакета, содержащего манифест, а за тем асинхронное его извлечение. Процесс получение запроса пакета, содержащего манифест, не определен в классе *ModelLoader* и реализуется в его дочерних классах, например на рисунке 18 показана реализация для локальной загрузки в классе *LocalModelLoader*.

После инициализации загрузчик может принимать запросы моделей, которые будет обрабатывать асинхронно, как изображено на схеме 19. Загрузка ресурсного пакета с целевой информационной моделью аналогична тому, как происходит загрузка манифеста: сначала происходит асинхронная загрузка нужного ресурсного пакета, а за тем асинхронное извлечение содержимого. Процесс получения запроса нужного ресурсного пакета, как и в случае с манифестом, не определен в классе *ModelLoader*, а потому реализуется в дочернем классе, как это показано на схеме 20. В отличие от

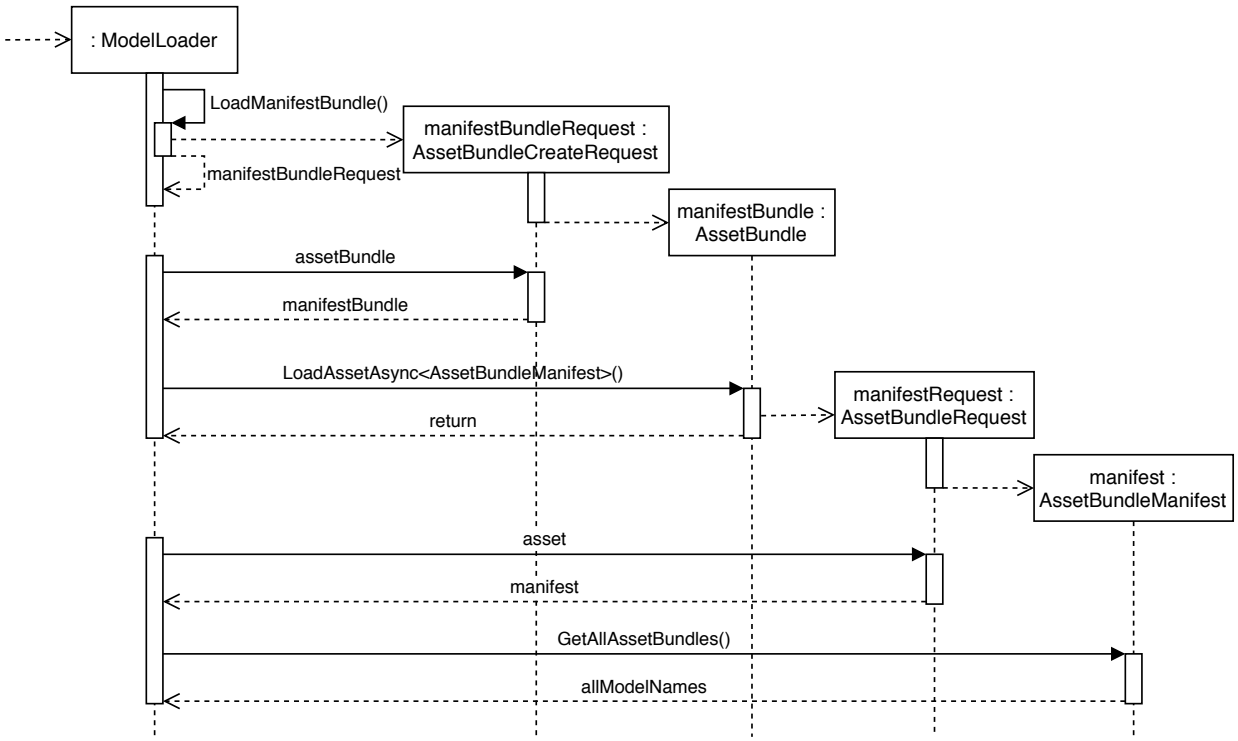

Рис. 17: Инициализация загрузчика моделей.

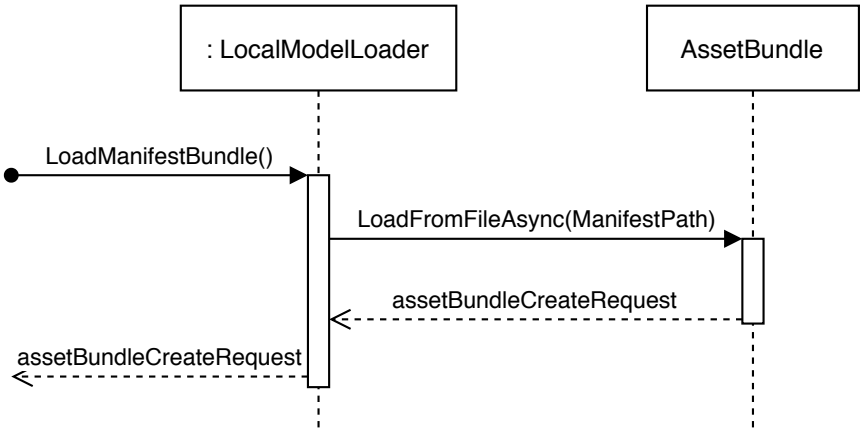

Рис. 18: Загрузка локально-расположенного манифеста пакетов.

загрузки манифеста теперь также требуется отслеживать наличие уже загруженных информационных моделей (*currentBundle*) для избежания перерасхода оперативной памяти. Еще одним отличием выступает необходимость возврата загруженной модели в асинхронном запросе, что реализовано через передачу в запрос функции обратного вызова (*onCompleteEvent*).

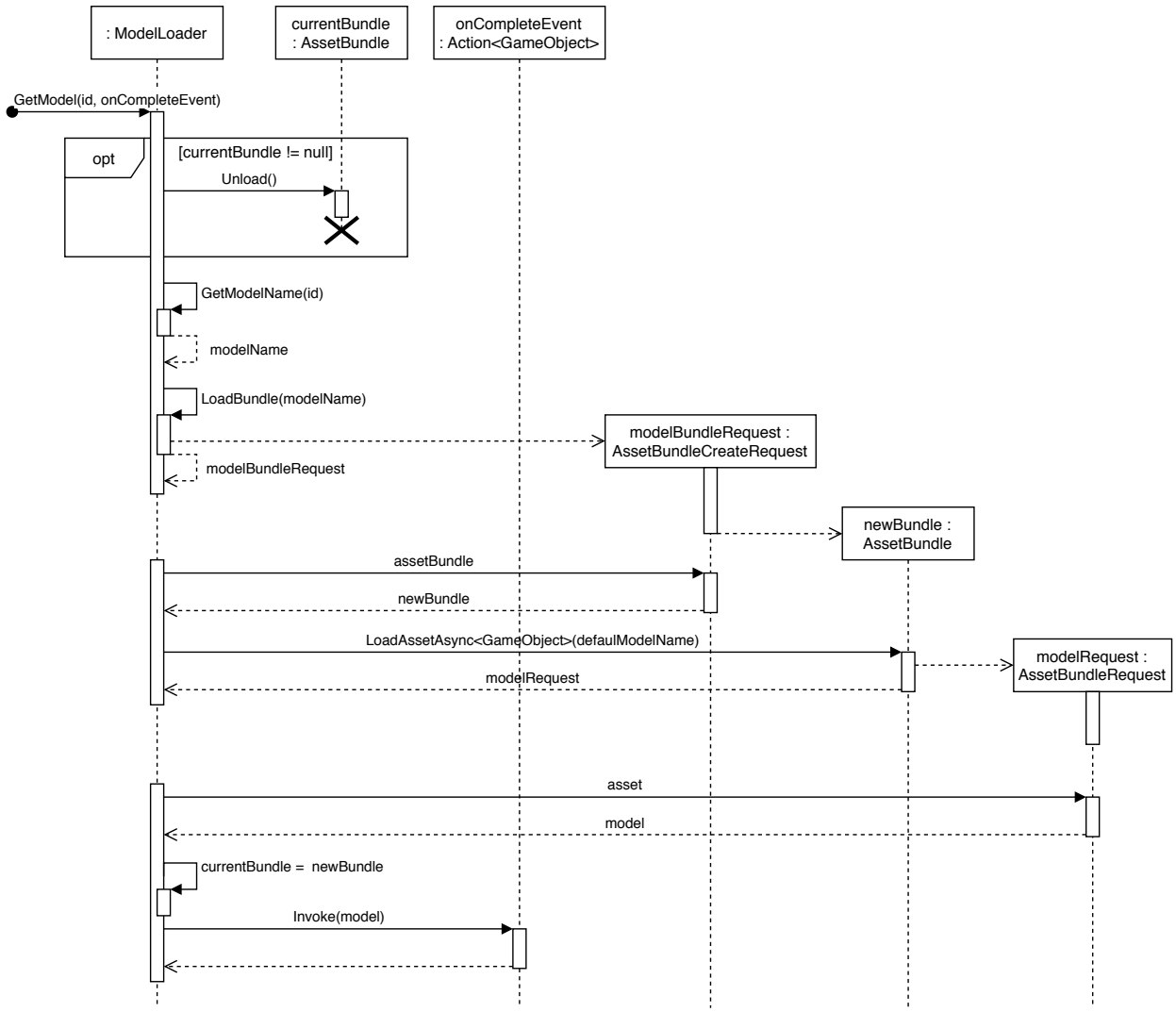

Рис. 19: Запрос загрузки модели.

Размещение модели

После загрузки и распаковки модель должна размещаться на виртуальном столе. Как было сказано в разделе 2.2, габариты размещаемой модели должны соответствовать доступному на столе пространству. В дополнение к этому пользователь должен иметь возможность изменять свой размер в зависимости от масштабов загруженной модели.

В компонентной архитектуре фреймворка Unity за расположение объектов в пространстве отвечает компонент *Transform*. Этот компонент обладает такими параметрами, как положение (*Position*) и размер (*Scale*) объекта, выражаемые трехмерными веще-

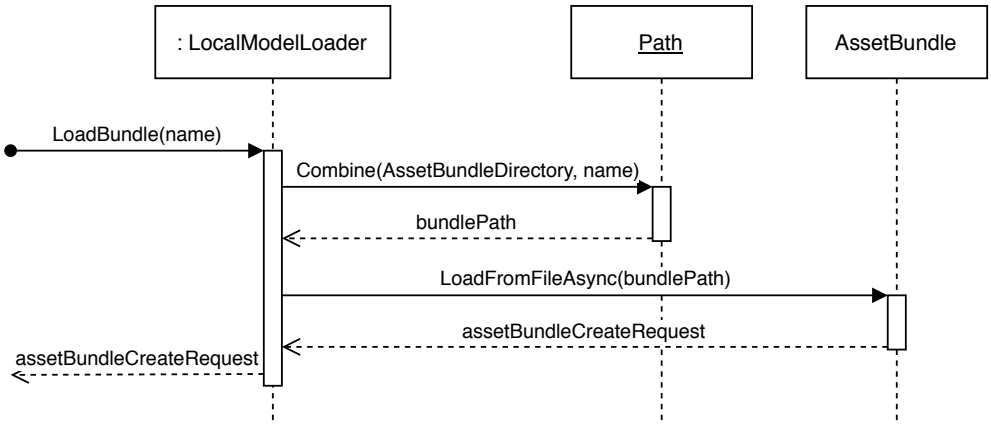

Рис. 20: Загрузка локально-расположенного ресурсного пакета.

ственными векторами (*Vector3*), а также поворот (*Rotation*), выражаемый кватернионом (*Quaternion*). Компонент *Transform* может исчислять свои параметры в глобальной системе координат, либо локально относительно другого компонента *Transform*. Компонент *Transform* добавляется абсолютно каждому объекту виртуальной сцены Unity при инстацировании объекта и используется многими стандартными модулями, например встроенными системами физики и отрисовки объектов.[4]

Для размещения модели реализуется вспомогательный компонент *Stand*, привязанный к объекту виртуального стола. Процесс расчета положения и размера для загруженной модели показан на рисунке 21.

При размещении новой информационной модели необходимо в первую очередь произвести ее инспирирование, так как при ее распаковке она не размещается внутри сцены, а только загружается в память. Также стоит учесть возможное наличие ранее загруженных моделей.

Для дальнейшего расчета размеров и положения размещаемой модели используются структуры *Bounds*, представляющие собой параллелепипеды, ограничивающие область пространства, в которой происходит отрисовка какого-либо трехмерного объекта. Эти структуры вычисляются как для размещаемой модели, так и для виртуального стола. После этого вычисляется вспомогательная величина *rescaleFactor* согласно формуле:

$$r = \frac{\min(sx, sz)}{\max(mx, mz)}$$

где r – *rescaleFactor*, sx и mx – габариты виртуального стола и информационной модели вдоль оси X, sz и mz – вдоль оси Z. В дальнейшем *rescaleFactor* используется для вычисления новых размера и положения модели, а также используется при изменении размеров пользователя. Формула расчета новых габаритов модели показана на схеме 21, а формула для расчета нового положения приведена ниже:

$$\vec{p} = \vec{sc} + \vec{up} * sy/2 + r * (\vec{up} * my/2 - \vec{mc})$$

где $\vec{p}$ – новое положение модели в пространстве (*newPosition*), sy и my – габариты вир-

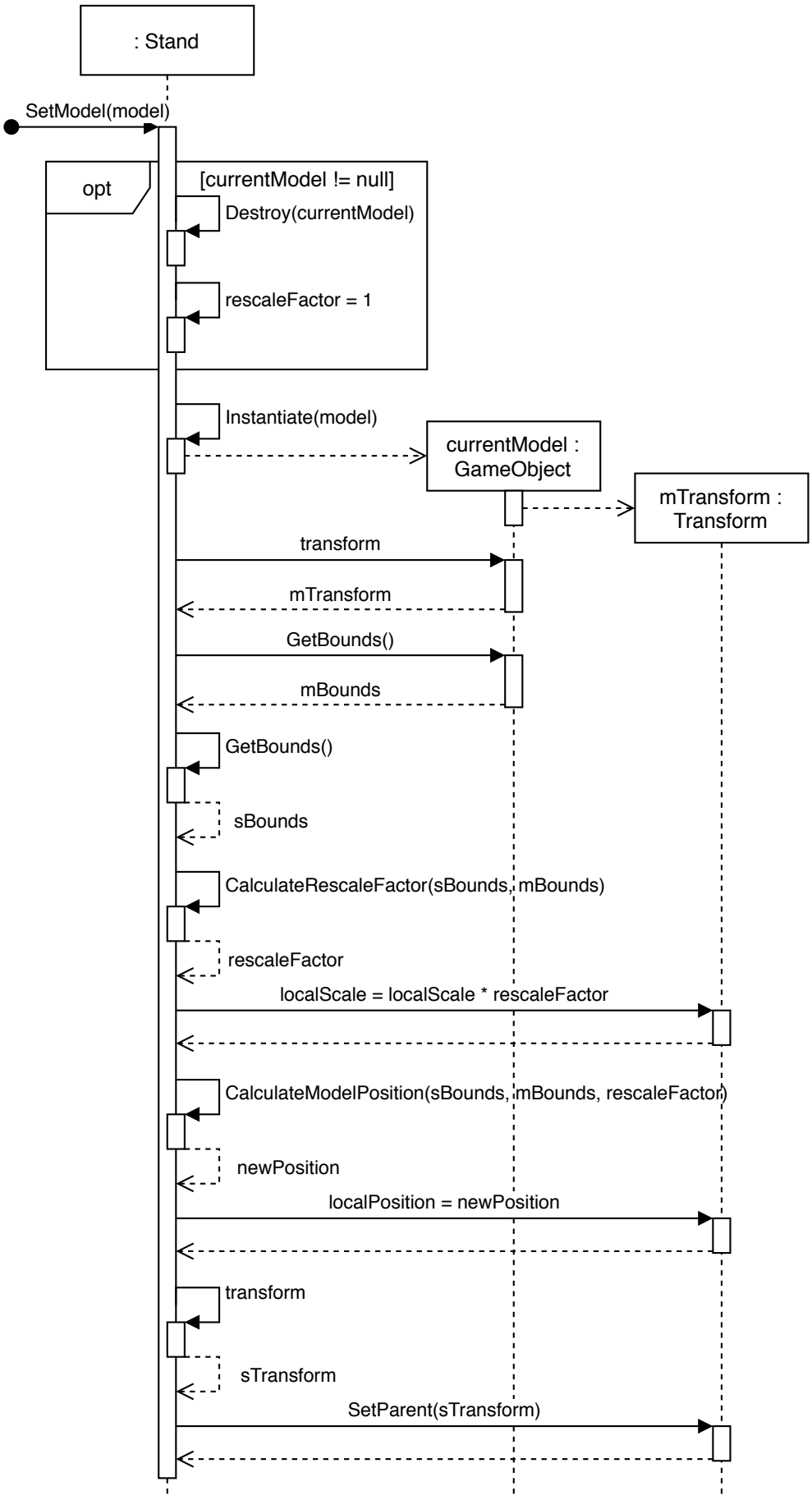

Рис. 21: Размещение модели на виртуальном столе.

туального стола и информационной модели вдоль оси Y, а $\vec{sc}$ и $\vec{mc}$ – их геометрические центры. В завершение компонент *Transform* размещаемой модели делается дочерним для *Transform*'а стола.

После размещения модели пользователь может изменять свой размер за счет вспомогательного компонента *UserScaler*, использующего параметр *rescaleFactor*, рассчитанный при размещении информационной модели. Изменение размера пользователя происходит в пределах между естественными размерами пользователя и размерами пользователя, соответствующими масштабу размещенной модели, что соответствует алгоритму на схеме 22.

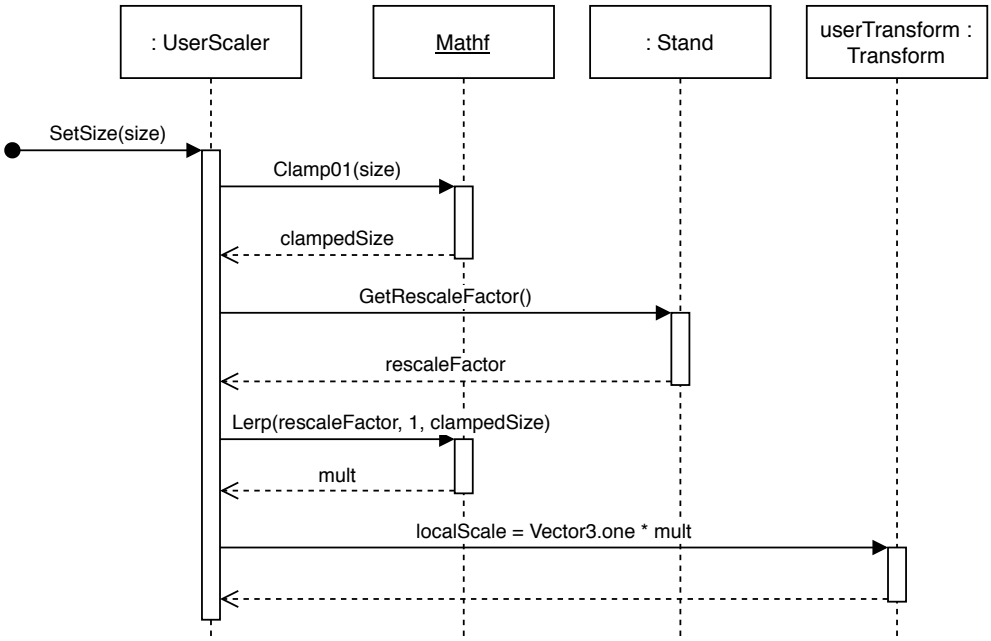

Рис. 22: Изменение размеров пользователя относительно габаритов модели.

Для изменения размеров используется вещественная величина (передаваемый параметр), которая ограничивается интервалом от 0 до 1. Для вычисления итогового размера пользователя проводится линейная интерполяция между *rescaleFactor*, вычисленным при установке модели, и 1 в точке, равной ограниченным входным параметром. Полученная величина умноженная на единичный вектор является новым размером пользователя. При передачу в функцию 0 размеры пользователя будут соответствовать масштабам размещенной модели, при передачи 1 – естественным размерам пользователя. Пример размещенной на виртуальном столе модели здания отеля показан на рисунке 23.

Слои модели

Наконец, после успешного размещения модель разбивается на слои, представляющие различные структурные элементы здания, например стены, окна или лестницы. На любой выделенный слой можно накладывать эффект отображения, как это описывалось в разделе 2.2. Для разбиения модели на слои реализуется вспомогательный

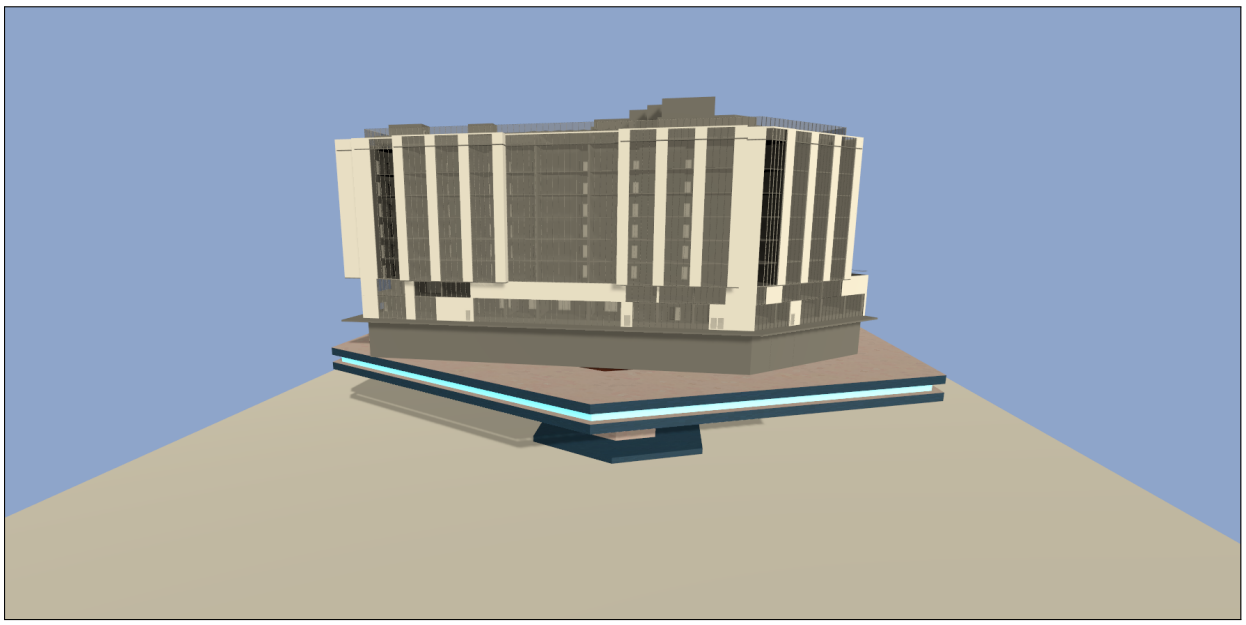

Рис. 23: Пример размещенной на виртуальном столе информационной модели здания.

класс *LayerController*, чей принцип работы показан на схеме 24.

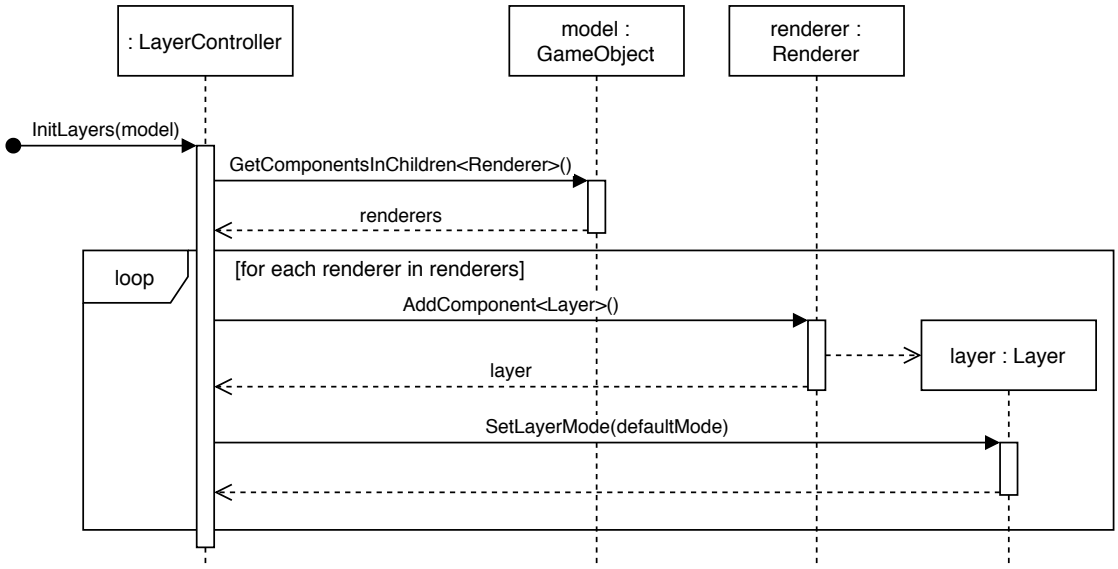

Рис. 24: Разбиение модели на слои.

Слои модели были реализованы с помощью компонента *Layer*. Для разбиения на слои можно воспользоваться тем фактом, что при экспорте модель проходила процесс статического батчинга (разделы 2.3 и 3.2), тем самым каждый потенциальный слой модели имеет свой компонент *Renderer*, отвечающий за отрисовку, и является дочерним по отношению к самой модели. Следовательно, каждый такой объект можно рассматривать как слой модели. После разбиения каждому слою модели может быть назначен свой режим отображения, как показано на рисунке 25.

В рамках прототипа предполагалось, что режимам отображения слоя достаточно иметь какое-либо поведение только при их добавлении и снятии, поэтому в текущей реализации реализованы только методы *OnSet* и *OnRemove*, хотя в дальнейшем их

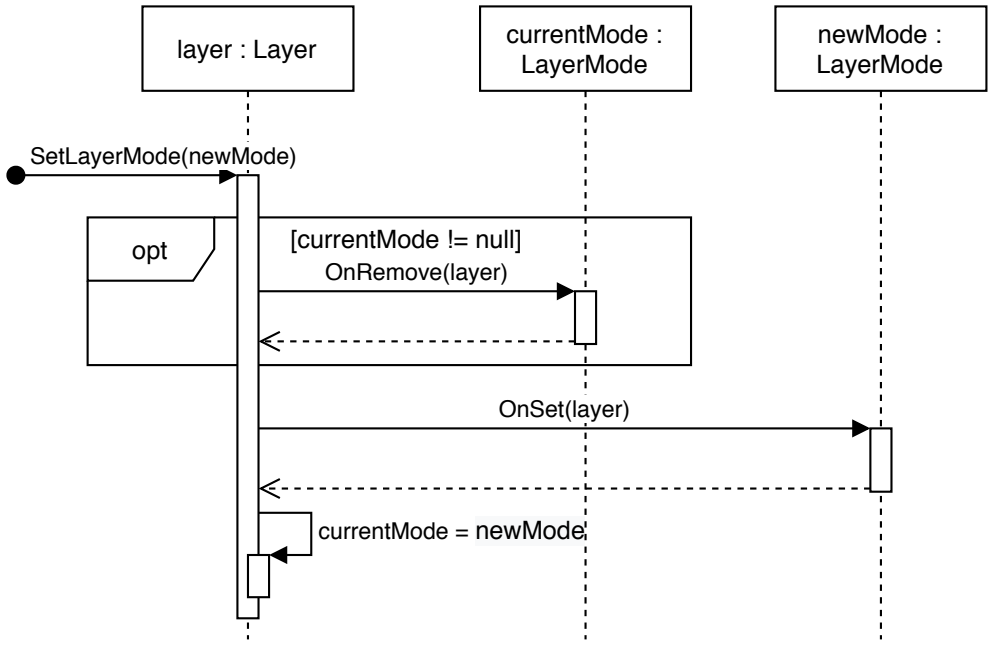

Рис. 25: Изменение режима отображения слоя модели.

набор может быть расширен. Всего было реализовано 4 режима отображения:

- “Стандартный”

  Стандартный режим не имеет какого-либо особого поведения и демонстрирует слой в том виде, в котором он изначально задумывался.

- “Невидимый”

  Невидимый режим отключает отображение через отключение активности слоя.

- “Срезка”

  Режим срезки позволяет отключать отрисовку части геометрии слоя за счет ее отсечения вспомогательной плоскостью. Режим реализован через замену шейдеров отрисовки трехмерной модели слоя на один из специально-написанных. Всего было написано 4 шейдера, имеющих одинаковую логику, но использующихся для разных способов отрисовки основанных на физических свойствах поверхности (metalic-specular[4]), а также разной степени прозрачности материала (непрозрачный-полупрозрачный). В основе шейдеров использовался принцип, при котором отрисовка модели прекращалась для тех участков, геометрия которых удовлетворяла формуле:

  $$(\overrightarrow{Pos} - \overrightarrow{plPos}) \bullet \overrightarrow{plNor} > 0$$

  где $\overrightarrow{plPos}$ и $\overrightarrow{plNor}$ – положение и нормаль отсекающей плоскости, а $\overrightarrow{Pos}$ – положение точки на поверхности модели.[25]

- “Подсветка”

Режим подсветки позволяет пользователю видеть скрытые за препятствиями части слоя за счет их обведения цветным контуром. Для реализации функциональности использовалась сторонняя библиотека с открытым исходным кодом.[26]

Пример работы режимов отображения можно увидеть на рисунке 26. В данном примере к слою лестниц был применен режим “Подсветка”, а к слоям стен, окон и пола – “Срезка”.

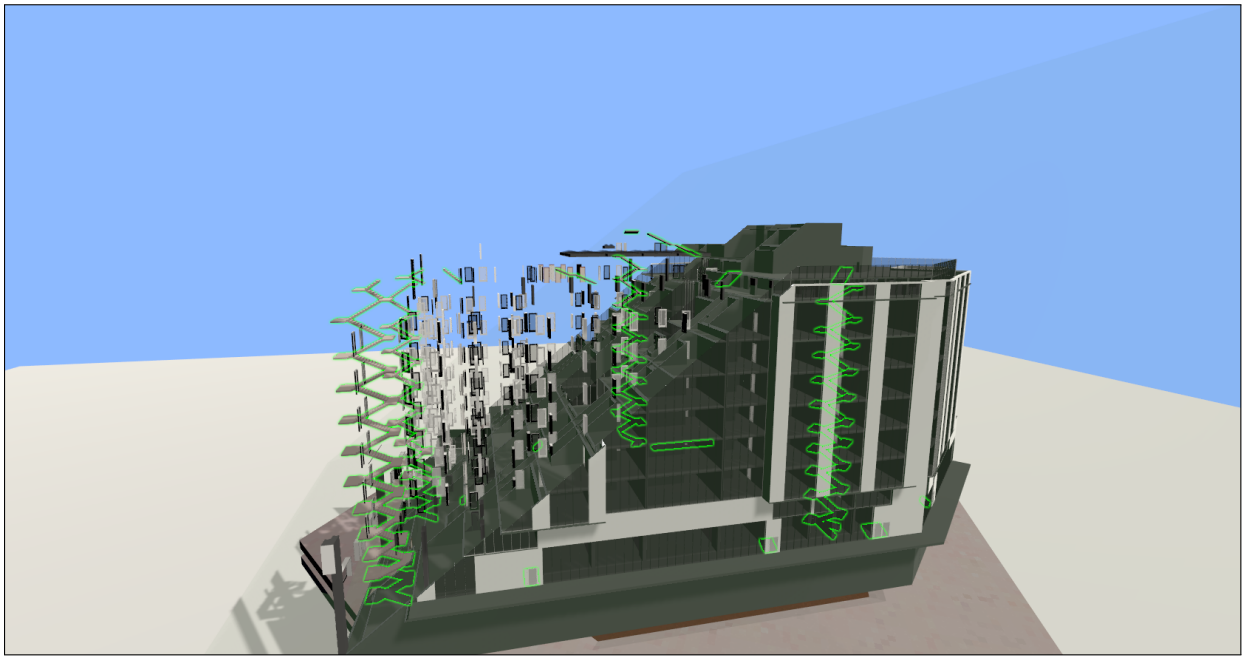

Рис. 26: Пример использования нескольких режимов отображения слоев.

Заключение

В ходе выполнения работы был разработан прототип приложения, позволяющего инспектировать информационные модели в среде виртуальной реальности. Все задачи проекта были выполнены, а именно:

1. реализован процесс извлечения атрибутивной информации модели на уровне категорий объектов, задаваемых в Revit;
2. реализованна автоматическая конверсия информационной модели из формата редактора Revit в пакет, загружаемый на клиентской части приложения;
3. реализована возможность манипулирования представлением информационной модели в клиентской части приложения.

Разработанный прототип в дальнейшем может быть модернизирован или полностью пересмотрен для добавления новой функциональности. На данный момент рассматривается возможность синхронизации изменений информационной модели в реальном времени без повторного конвертирования, что потребует значительного пересмотра работы серверной части приложения. Помимо этого проводятся исследования по повышению качества извлечения атрибутивной информации модели.

Список литературы

1. *Johnson R.*, *Foote B.* Designing Reusable Classes // Journal of Object-Oriented Programming. — 1988. — июнь. — т. 1. — с. 22—35.
2. *Schmidt D.* Applying Design Patterns and Frameworks to Develop Object-Oriented Communication Software. — 2000. — апр.
3. *National Institute of Building Sciences.* Frequently asked questions about the National BIM Standart - United States. — URL: `https://www.nationalbimstandard.org/faqs` (дата обр. 01.02.2020).
4. *Unity Technologies.* Unity User Manual. — URL: `http://docs.unity3d.com` (дата обр. 05.04.2020).
5. *Путин В.* Перечень поручений Президента Российской Федерации по итогам заседания Государственного совета Российской Федерации 17 мая 2016 г. Пр-1138 ГС. — URL: `https://tomsk.gov.ru/uploads/ckfinder//userfiles/files/%D0%9F%D1%80-1138%D0%B3%D1%81.PDF` (дата обр. 08.04.2020).
6. The Past, Present, and Future of Virtual and Augmented Reality Research: A Network and Cluster Analysis of the Literature / P. Cipresso [и др.] // Frontiers in Psychology. — 2018. — нояб. — т. 9.
7. *Akpan I. J.*, *Shanker M.* A comparative evaluation of the effectiveness of virtual reality, 3D visualization and 2D visual interactive simulation: an exploratory meta-analysis // SIMULATION. — 2018. — февр. — т. 95, № 2. — с. 145—170.
8. *Unity Technologies.* Unity Reflect. — URL: `https://unity.com/products/reflect` (дата обр. 15.02.2020).
9. *IrisVR.* VR for Architecture, Engineering, and Construction. — URL: `https://irisvr.com/` (дата обр. 17.02.2020).
10. *Enscape.* Enscape - Architectural Visualization Software for Revit, SketchUp, Rhino & ArchiCad. — URL: `https://enscape3d.com/` (дата обр. 17.02.2020).
11. Integration of VR with BIM to facilitate real-time creation of bill of quantities during the design phase: a proof of concept study / J. Davidson [и др.] // Frontiers of Engineering Management. — 2019. — июнь.
12. OpenBIM-Tango integrated virtual showroom for offsite manufactured production of self-build housing / F. P. Rahimian [и др.] // Automation in Construction. — 2019. — т. 102. — с. 1—16. — ISSN 0926-5805.
13. *BuildingSmart.* Industry Foundation Classes (IFC). — URL: `https://www.buildingsmart.org/standards/bsi-standards/industry-foundation-classes/` (дата обр. 16.04.2020).
14. *Autodesk Inc.* Welcome to Revit 2019 Learning. — URL: `http://help.autodesk.com/view/RVT/2019/ENU/` (дата обр. 20.01.2020).

15. *Weech S., Kenny S., Barnett-Cowan M.* Presence and Cybersickness in Virtual Reality Are Negatively Related: A Review // Frontiers in Psychology. — 2019. — февр. — т. 10.

16. *Cohen-Or D., Chrysanthou Y., Silva C.* A Survey of Visibility for Walkthrough Applications // Proceedings of SIGGRAPH. — 2001. — янв.

17. *Liptak A.* How Neill Blomkamp and Unity are shaping the future of filmmaking with Adam: The Mirror. — URL: https://www.theverge.com/2017/10/4/16409734/unity-neill-blomkamp-oats-studios-mirror-cinemachine-short-film (дата обр. 15.04.2020).

18. *Edelstein S.* How gaming company Unity is driving automakers toward virtual reality. — URL: https://www.digitaltrends.com/cars/unity-automotive-virtual-reality-and-hmi/ (дата обр. 15.04.2020).

19. *Captain S.* How Google's DeepMind will train its AI inside Unity's video game worlds. — URL: https://www.fastcompany.com/90240010/deepminds-ai-will-learn-inside-unitys-video-game-worlds (дата обр. 15.04.2020).

20. *Microsoft.* C# documentation. — URL: https://docs.microsoft.com/en-us/dotnet/csharp/ (дата обр. 07.03.2020).

21. *Valve Corporation.* SteamVR Unity Plugin. — URL: https://valvesoftware.github.io/steamvr_unity_plugin/ (дата обр. 03.12.2019).

22. *Wikipedia, the free encyclopedia.* HTC Vive. — URL: https://en.wikipedia.org/wiki/HTC_Vive (дата обр. 03.02.2020).

23. *Autodesk Inc.* 3ds Max Learning Center. — URL: https://help.autodesk.com/view/3DSMAX/2019/ENU/ (дата обр. 07.03.2020).

24. *Unity Technologies.* Assets, Resources and AssetBundles. — URL: https://learn.unity.com/tutorial/assets-resources-and-assetbundles (дата обр. 15.05.2020).

25. *Aldandarawy A.* Unity3d Cross Section Shader Using Shader Graph. — URL: https://learn.unity.com/tutorial/assets-resources-and-assetbundles (дата обр. 21.01.2020).

26. *Arvtesh.* UnityFx.Outline. — вер. 0.7.0. — 27.11.2019. — URL: https://github.com/Arvtesh/UnityFx.Outline.