Оглавление

Введение

1. Обзор существующих решений

1.1 3D тур по Санкт-Петербургскому горному университету

1.2 Панорама Boeing-747. Оренбургские авиалинии

1.3 Интерактивная 3D презентация гостиницы «Галакт»

1.4 Сравнение аналогов

2. Анализ средств разработки

3. Постановка задачи

4. Разработка приложения

4.1 Проектирование

4.2 Архитектура приложения

4.3 Разработка кода

4.3.1 Настройка подгружаемых ресурсов

4.3.2 Покадровый рендеринг графики и шаблон наблюдатель

4.3.3 Создание сцены, камеры и порта просмотра

4.3.4 Загрузка и установка заданных ресурсов

4.3.5 Выделение моделей на сцене, управление камерой

4.3.6 Сохранение состояния программы

4.4 Разработка пользовательского интерфейса

4.4.1 Интерфейс главного экрана

4.4.2 Меню настроек

4.5 Тестирование

5. Разработка руководства пользователя

Заключение

Список используемой литературы

Приложение

# Введение

В настоящее время системы 3D визуализации активно используются в таких сферах деятельности, как телекоммуникации, ракетостроение, медицина и т.д. Однако такие системы чаще всего являются сложными и привязанными к конкретной реализации. В работе рассматривается более универсальный подход к проектированию и реализации системы 3D визуализации. Полученную при помощи такого подхода реализацию планируется использовать во время профориентации на кафедре АВТ, таким образом целевой аудиторией являются абитуриенты.

Система визуализации – это сложная, продуманная структура для предоставления информации пользователю. В современном мире актуальна проблема подачи информации аудитории, так как в условиях огромного количества информации сложно привлечь внимание аудитории и предоставить информацию на понятном пользователю языке.

Цель дипломной работы – разработка универсальной системы визуализации. Для достижения поставленной цели необходимо решить следующие задачи:

1. Исследовать существующие системы визуализации
2. Исследовать подходящие библиотеки для разработки 3D сцены
3. Реализовать возможность загрузки и отображения множества объектов на сцене
4. Реализовать и протестировать систему визуализации

В работе отражена разработка приложения, которая интерактивно предоставляет слушателю информацию о выбранных в ходе презентации объектах.

# 1. Обзор существующих решений

Прежде чем определить дальнейшее видение проекта и требования к нему, нужно проанализировать существующие решения.

Существуют системы визуализации нескольких видов:

* системы, использующие технологии панорамной сферической съёмки;
* системы, использующие рендеринг трехмерной графики

Для сравнительного анализа были выбраны три системы-аналога. Далее рассмотрим их особенности.

## 3D тур по Санкт-Петербургскому горному университету

Данная система позволяет детально увидеть Петербургский университет изнутри при помощи технологии панорамной сферической съёмки [5]. Это камеры, снимающие видео на 360 градусов. Из полученных записей составляются виртуальные панорамы в соответствии с рисунком 1.

Виртуальные панорамы предназначены для показа на компьютере с помощью специального программного обеспечения, позволяющего зрителю «крутить головой», глядя на разные части пространства, окружающего фотографа при съёмке. В основе сферической фотографии лежит собранное из множества отдельных кадров изображение в сферической проекции. Фрагменты таких фотографий склеиваются и далее, при помощи Flash-технологий, разрабатывается виртуальный 3D тур.

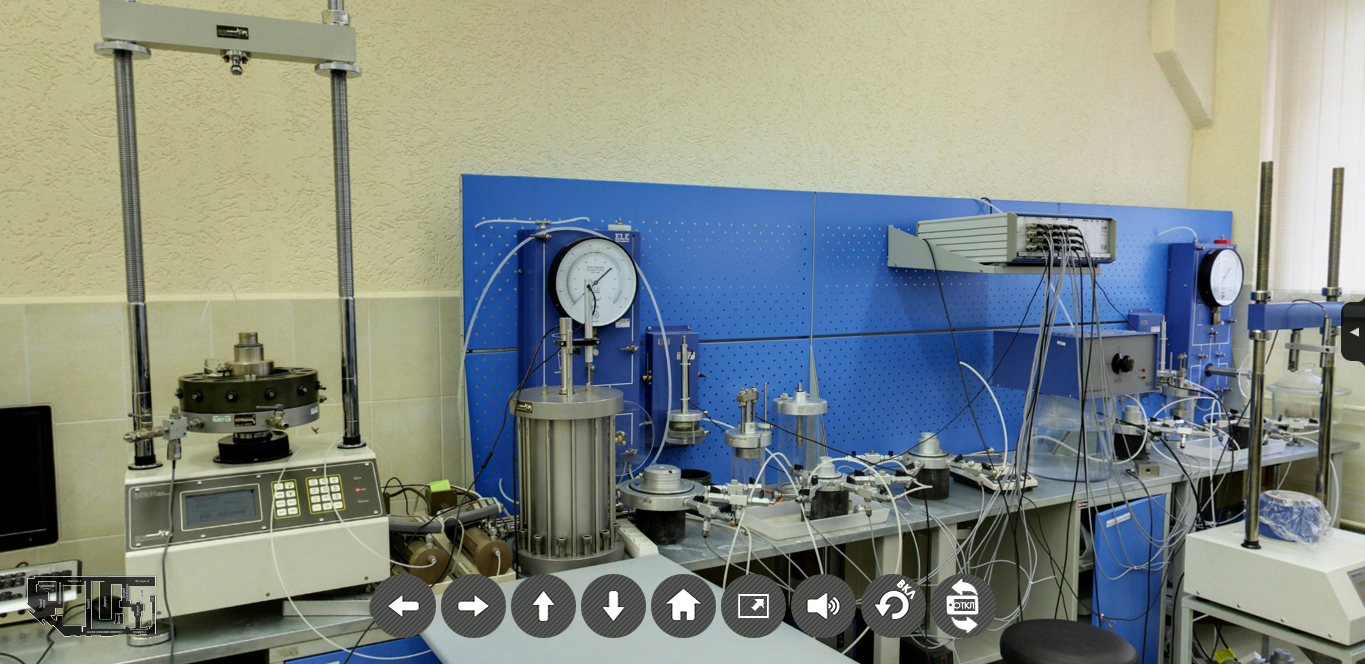


Рисунок 1 – Пример панорамной системы визуализации

## Панорама Boeing-747. Оренбургские авиалинии

Панорама работает по аналогии с предыдущим аналогом, используя виртуальные панорамы в качестве предоставления информации. Отличительной чертой является наличие информации при переходе между сценами в соответствии с рисунком 2. Панорама устроена следующим образом: в ней реализовано несколько сцен, которые представлены одной виртуальной панорамой, имеющие уникальное описание [6].



Рисунок 2 – Пример панорамной визуализации

## Интерактивная 3D презентация гостиницы «Галакт»

Для представления информации презентация использует рендеринг трёхмерной графики в соответствии с рисунком 3. Интерактивная связь с пользователем осуществляется при помощи мыши. Пользователь свободно перемещается по виртуальному миру, взаимодействуя с различными объектами, которые предоставляют информацию о себе при взаимодействии [7].

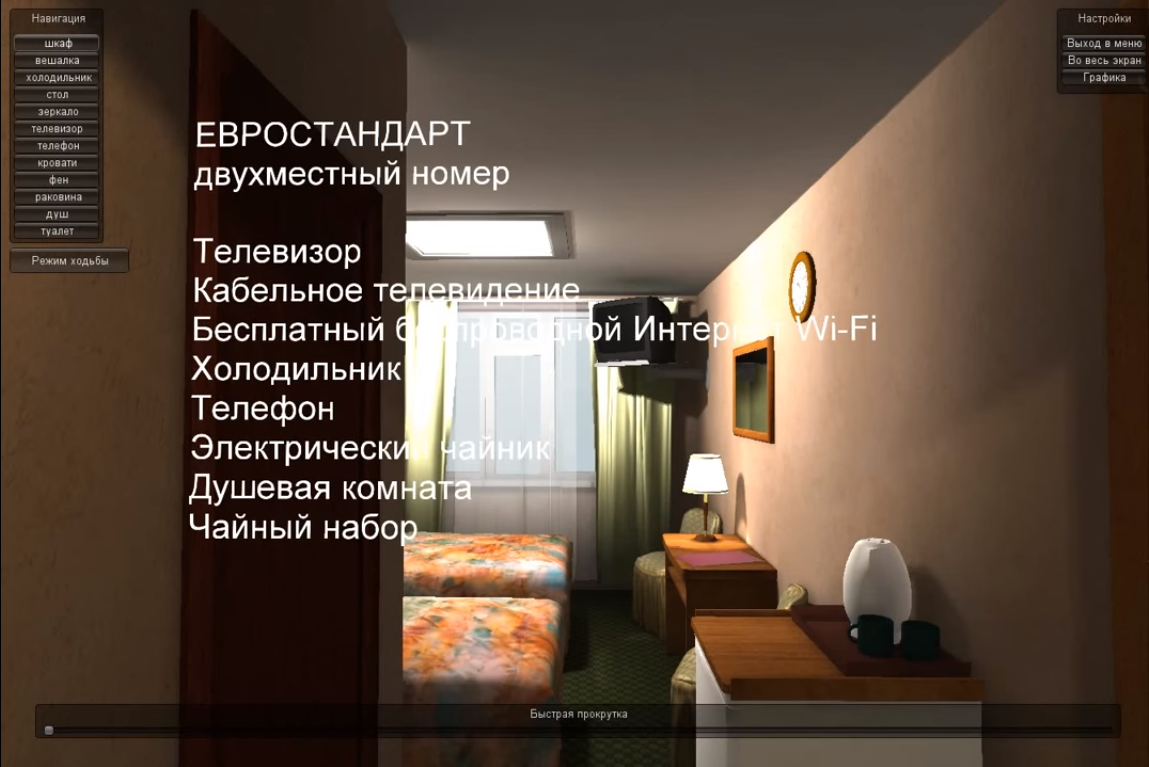


Рисунок 3 – Пример системы визуализации с применением рендера

## Сравнение аналогов

Схожесть рассмотренных систем заключается в детальном представлении сцены с различных состояний панорамной или виртуальной камеры.

Различаются системы:

* наличием интерактивной связи с пользователем
* видом представления информации

В отличие от 3D тура, интерактивная презентация способна выполнять следующие задачи:

* подгрузка собственных ресурсов
* реализация интерактивной связи
* меньшая нагрузка на CPU

Так как в проекте предусматривается интерактивная связь с пользователем и внедрение анимации, чтобы обратная связь выглядела ещё более информативно, интерактивная презентация рассматривается в качестве основного аналога.

# 2. Анализ средств разработки

Важным этапом в разработке системы визуализации является выбор средств разработки. Это может повлиять на такие свойства системы как структурность и функциональность.

Существует множество библиотек, которые отличаются своей сложностью, направленностью, архитектурой построения. Одними из самых распространённых являются Ogre 3D, UnrealEngine и OpenSceneGraph.

OGRE (Object-Oriented Graphics Rendering Engine) – объектно-ориентированный графический движок с открытым исходным кодом, написанный на C++. Автором OGRE является Стив Стриитинг (англ. Steve Streeting). интерфейс рендеринг программный

Ogre 3D является графическим движком для рендеринга трехмерной графики. Большую популярность движок получил за счет своей гибкости, что позволяет интегрировать его со многими другими библиотеками (физика – ODE, Newton, PhysX, Bullet; звук, сеть, графический интерфейс и т. д.).

Для реализации графического интерфейса пользователя (англ. GUI – Graphical User Interface) могут применяться как стандартные функции графического интерфейса OGRE, так и импортироваться сторонние библиотеки (OpenGUI, MyGUI).

Ogre 3D обладает следующими возможностями: поддержка платформ Windows, Linux и Mac OS X, Android, iOS; скриптовая система управления материалами (мультитекстурирование, мультипроходное смешивание); экспортеры для основных коммерческих и свободных пакетов 3D моделирования; система управления ресурсами; поддержка Direct3D, OpenGL; сложная скелетная анимация (анимация тела), анимация гибких форм, морфинг (анимация лица), анимация пути (камера, перемещение) [1].

UnrealEngine – игровой движок, написанный на языке С++, разрабатываемый и поддерживаемый компанией EpicGames. Он позволяет создавать игры для большинства операционных систем и платформ, а также для различных портативных устройств. Например, для устройств Apple (iPad, iPhone), управляемых системой iOS. Впервые работа с iOS была представлена в 2009 году. В 2010 году продемонстрирована работа движка на устройстве с системой webOS.

Для упрощения портирования движок использует модульную систему зависимых компонентов, а также поддерживает различные системы рендеринга: Direct3D, OpenGL, S3, PowerVR; воспроизведения звука: EAX, OpenAL, DirectSound3D; средства голосового воспроизведения текста, распознавание речи, модули для работы с сетью и поддержки различных устройств ввода[3].

OpenSceneGraph – это открытое программное обеспечение для разработки высокопроизводительных 3D приложений, используемое разработчиками для компьютерных игр, программ виртуальной реальности, компьютерного моделирования.

Инструментарий написан на C++ с использованием библиотек OpenGL. OpenSceneGraph поддерживает большинство популярных операционных систем: Windows, Mac OS X, Linux, IRIX, Solaris и FreeBSD.

В особенности библиотеки входит: полнофункциональная поддержка промышленной графической сцены; поддержка большого количества 2D изображений и 3D форматов, благодаря 34 импортеров из таких индустриальных форматов как OpenFlight, TerraPage, OBJ, 3DS, JPEG, PNG и GeoTIFF; система частиц; поддержка кадрового буфера, пиксельного буфера и буфера рендеринга в текстуры; поддержка библиотек, позволяющих внешне воздействовать на приложение, выполнять запросы на получение и передачу данных и оперировать всеми классами в графической сцене[2].

При выборе движка для реализации проекта нужно провести анализ их качеств и выбрать оптимальный вариант. Можно выделить следующие критерии выбора:

* Открытый исходный код – возможность использовать готовый исходный код движка для просмотра, редактирования, изучения.
* Поддержка операционных систем – возможность запуска готового приложения под определённой платформой.
* Поддержка различных 3D API – возможность интегрировать интерфейсы программирования приложений.
* Распространённость – широкая область применения в разработке 3D приложений.
* Коммерция – возможность коммерческой реализации готового продукта.
* Community – доступность документации для изучения.

Соответствие рассматриваемых движков выбранным критериями представлено в таблице 1.

Таблица 1 – Критерии выбора 3D библиотеки для разработки

|  |  |  |  |
| --- | --- | --- | --- |
| Критерии | Ogre 3D | UnrealEngine | OpenSceneGraph |
| Открытый исходный код | + | - | + |
| Поддержка ОС Windows, Mac OS, Linux, iOS, Android | + | + | + |
| Поддержка различных 3D API (DirectX, OpenGL) | + | + | - |
| Распространённость | + | + | - |
| Коммерция | + | - | + |
| Community | + | + | +/- |

На основе результатов анализа качеств 3D движков для программной реализации проекта в образовательных целях можно сделать вывод, что оптимальной средой разработки является Ogre 3D.

Ogre 3D представляет собой объектно-ориентированный фреймворк, который включает все части процесса рендеринга в объектной модели. С объектно-ориентированным подходом к рендерингу геометрии, необходимость во взаимодействии с ней напрямую совершенно отпадает.

Используется версия библиотеки Ogre SDK 1.81 – наиболее стабильная версия движка.

Исходя из того, что библиотека Ogre3D написана на языке С++, языком разработки приложения является С++. Наиболее удобной средой разработки под С++ является Visual Studio. Так как выбранная версия библиотеки успешно компилируется на компиляторе версии Visual Studio 2010, используется компилятор версии 100 под обёрткой Visual Studio 2012.

Далее необходимо выбрать стороннюю графическую библиотеку для удобства разработки graphical user interface (GUI). Наиболее распространённой и практичной библиотекой, которая работает с Ogre3D, является CEGUI (Crazy Eddie’s GUI). Используется версия CEGUI 0.79, так как она успешно интегрируется с выбранной версией движка.

Для разработки 3D моделей и их анимаций используется Blender 3D, так как Ogre имеет ресурс для экспорта (blender2ogre) 3D моделей с дальнейшей возможностью использования в движке. Исходя из совместимости экспортёра с версией Blender 0.79, используется данная версия.

В итоге, для разработки используются следующие средства:

* Ogre SDK 1.81
* Visual Studio 2012 (С++)
* CEGUI 0.79
* Blender 3D version 2.66
* Blender2ogre 0.6 (export)

# 3. Постановка задачи

Аудитории зачастую нужно получить информацию в понятном и структурированном виде. Для достижения этих целей разрабатываются различные виды презентаций, виртуальных туров или иных систем визуализаций.

Информация, поданная в виде электронных компьютерных презентаций, легко усваивается конечным потребителем. Такие презентации незаменимы на конференциях и всевозможных защитах проектов, это отличный способ анонсировать новую информацию, продукт, товар, услугу. Интерактивная презентация – самый быстрый и современный способ донести любую информацию.

Презентацию можно представить с элементами видео или картинок, но больше информации можно донести при помощи 3D сцены. Информация, донесенная 3D сценой, будет более полной и детальной, что позволит в трёхмерном представлении обратить внимание аудитории на ключевые моменты в презентации.

В целях реализации приложения используется движок для рендера трёхмерной графики Ogre3D, использующий 3D сцену в качестве интерактивной связи с пользователем.

Нужно подробно предоставить информацию аудитории о каких-либо объектах, и наша задача состоит во внедрении интерактивной связи с элементами сцены в 3D презентацию. Под интерактивностью понимается способность системы к взаимодействию с пользователем, в нашем случае требуется добавить в функциональность сцены обратную связь с объектами сцены.

Рендеринг трёхмерной графики нагружает графический и центральный процессор, это нужно учесть и реализовать альтернативные настройки в интерфейсе сцены, позволяющие ослабить нагрузку на процессор. Среди настроек могут оказаться: отключение теней у объектов, а также настройка глобального освещения сцены, переключение на точечные источники света.

Таким образом, при разработке приложения нужно учитывать производительность слабых процессоров. В этих целях после завершения разработки, необходимо провести тестирование приложения на предмет производительности на различных операционных системах и различных видах процессоров.

# 4. Разработка приложения

Процесс разработки является ключевым и включает в себя несколько взаимосвязанных этапов:

* Проектирование – на данном этапе анализируются и подробно описываются задачи для дальнейшей разработки, удовлетворяющие целям проекта.
* Разработка – написание кода, удовлетворяющего требованиям проектной документации, реализованной на этапе проектирования.
* Тестирование – процесс исследования приложения на предмет обнаружения ошибок и их своевременное исправление.

После отладки в процессе тестирования приложение будет готово к эксплуатации и сопровождению.

Также требуется реализовать руководство пользователя для обеспечения возможности самостоятельного освоения и применения программы.

## 4.1 Проектирование

Важным этапом разработки является проектирование программной логики приложения. Проектирование требуется для определения этапов разработки и дальнейшего видения проекта.

Разработка разделена на несколько этапов:

1. Разработка программной логики представления 3D моделей на сцене

В первую очередь необходимо настроить сцену, как ключевой элемент для дальнейшей разработки. После чего требуется запрограммировать базовый функционал, включающий в себя выделение любой из загруженных 3D моделей, находящейся на сцене. После настройки выделения 3D модели, нужно реализовать движение камеры таким образом, чтобы при выделении объекта, она приближалась к нему на заданное расстояние в соответствии с рисунком 4.

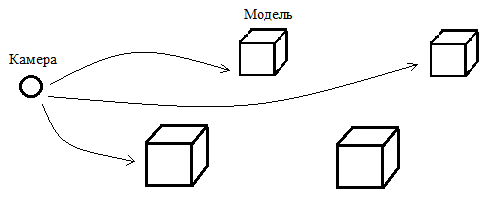


Рисунок 4 – Схема базового функционала приложения

После остановки камеры, запускается анимация 3D модели, одновременно с запуском анимации рядом выводится информация о 3D модели, где информационный текст зависит от проигрывающейся анимации.

Так как требуется реализовать возможность подгрузки и рендера множества объектов на сцене, нужно подгружать требуемые ресурсы из директории в виде zip архива с моделью, затем задать расположение всех этих моделей на сцене.

Также необходимо заранее продумать сохранение данных программы после её закрытия, так как планируется предоставить пользователю возможность манипулирования моделями на сцене. Для сохранения данных программы можно использовать текстовый xml файл. Так как движок не поддерживает работу с xml файлами, требуется выбрать стороннюю библиотеку для работы с ними.

Популярными библиотеками являются TinyXML и RapidXML – оба являются синтаксическими анализаторами файла. Первый минимален и легко интегрируется в различные программы, а второй наиболее быстрый и содержит большое количество функций. В итоге, для работы с xml файлом потребуется минимальное количество функций. Так как сохранять данные требуется только при выходе и при нажатии кнопки, то быстродействием можно пренебречь, поэтому выбрана библиотека TinyXML.

Разработка 3D моделей

При создании трехмерных моделей в целях использования в движке для рендера, требуются знания особенностей их моделирования. Движки используют различную топологию в 3D моделях: многоугольные полигоны, треугольные полигоны и другие. Ogre3D использует треугольные полигоны.

Важным аспектом при разработке 3D модели является количество полигонов, которые она содержит. Каждый полигон требует от графического процессора время на расчёты и на рендер этого полигона, поэтому стоит моделировать таким образом, чтобы количество полигонов было минимальным. Однако модель может получиться угловатой и потребуется дополнительное сглаживание. При использовании сглаживания увеличивается количество полигонов в топологии модели, поэтому рекомендуется использовать две или три группы сглаживания.

Для нашего проекта требуется реализовать интерьер в виде комнаты, в котором будут располагаться будущие 3D модели. Интерьер не требует большого количества полигонов, поэтому стоит реализовать его с цельной сеткой, так как цельный меш, имеющий большое количество полигонов, меньше нагружает сцену, чем раздельные меши.

Разработка элементов пользовательского интерфейса.

Интерфейс – важная часть любой программы, так как именно он является границей между программным функционалом и конечным пользователем, с помощью которого происходит взаимодействие и манипулирование программой.

Интерфейс разрабатывается при помощи сторонней графической библиотеки CEGUI и отвечает за связь с пользователем, а также за меню настроек, которое отвечает за настройку сцены и моделей.

Меню настроек должно включать в себя альтернативные настройки сцены в целях повышения производительности приложения, в качестве таких настроек могут быть: настройка теней и глобального освещения. Учитывая то, что система реализуется по принципу масштабируемости (неограниченное количество подгружаемых моделей), нужно также предусмотреть масштабируемость комнаты, в которой располагаются модели, то есть требуется регулировать масштаб осей с помощью интерфейса.

Чтобы была возможность расставить подгруженные модели в комнате, нужно реализовать меню манипулирования моделями в пространстве.

Так как выбранная графическая библиотека не поддерживает кириллическую кодировку по умолчанию, требуется реализовать декодер для русского алфавита.

Требуется выводить информацию в виде текста при выборе модели, поэтому нужно реализовать загрузку текста из файла, чтобы пользователь смог отредактировать или создать новый информационный текст для модели.

Тестирование

В целях отладки программного кода и выявления ошибок требуется провести тестирование на двух этапах разработки:

* Первый этап тестирования – после реализации базового функционала сцены
* Второй этап тестирования состоит из нескольких тест-кейсов:
  + тестирование производительности
  + проверка эргономичности (юзабилити-тестирование)

Исходя из описанных этапов проектирования, разработана диаграмма классов в соответствии с рисунком 5, по которой требуется реализовать программную часть приложения. Диаграмма направлена на представление статической структуры модели разрабатываемой системы, а также на определение типов классов и связей между ними, которые планируется использовать в программном коде.

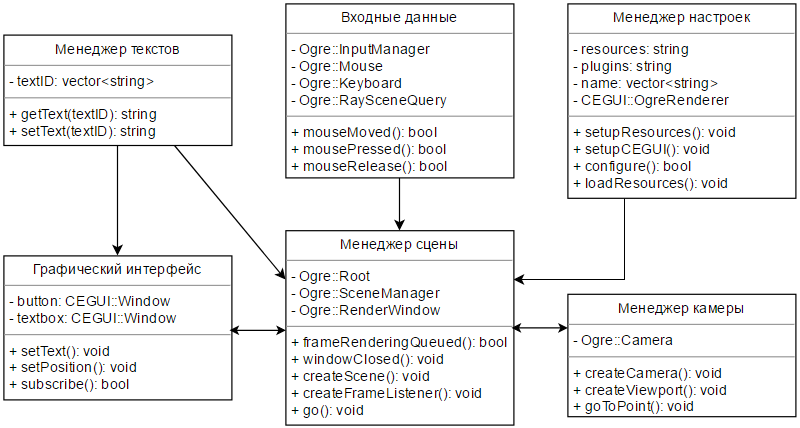


Рисунок 5 – Диаграмма классов приложения

Из диаграммы видно, что реализовано несколько классов, ключевым из которых является менеджер сцены, в котором происходит настройка всех параметров сцены. Также в отдельные классы вынесен графический интерфейс, загрузка текстов и прочие настройки.

## 

## 4.2 Архитектура приложения

3D сцена в библиотеке Ogre 3D имеет древовидную структуру, корневым элементом которой является главный узел сцены (RootSceneNode). Каждый узел сцены может иметь своих потомков (SceneNode). Таким образом, все сцены привязаны к главному узлу сцены.

К нескольким потомкам главного узла сцены необходимо привязать сцену в виде комнаты, в которой будут закреплены определённые 3D модели.

Приложение универсально в использовании, так как самостоятельно подгружает заданные пользователем параметры для каждой из комнат (3D модель и её описание) и отображает их на сцене. Таким образом, пользователь может менять параметры для различных комнат.

Архитектура приложения, в соответствии с рисунком 6, состоит из следующих классов:

Менеджер сцены – ключевой класс, отвечающий за рендер сцены, настройки расположения 3D моделей на сцене и анимации 3D моделей.

Входные данные – класс, который отвечает за считывание входных данных пользователя: обработка клика мышки для определения попадания луча (экземпляр класса Ray) на 3D модель, обработка клавиатуры.

Менеджер текстов – класс, в котором хранятся все тексты приложения. Также в нём реализована подгрузка текста с описанием 3D модели из файла.

Менеджер настроек – класс, отвечающий за конфигурацию приложения, подгрузку ресурсов приложения и за настройку сторонней графической библиотеки.

Графический интерфейс – отвечает за настройку графического интерфейса пользователя, а также за графические эффекты.

Менеджер камеры – менеджер настроек движения камеры.

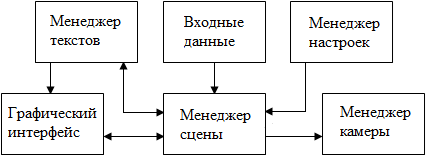


Рисунок 6 – Архитектура приложения

При разработке архитектуры приложения используются следующие шаблоны проектирования:

Faсade – применяется для абстрагирования классов графического интерфейса, менеджера настроек, менеджера сцены, входных данных.

Singleton – для обеспечения глобального доступа к экземплярам классов менеджера текстов, менеджера сцены, менеджера камеры.

Listener – для обеспечения взаимодействия классов. Реализуется интерфейс с помощью которого шаблон получает оповещения от классов входных данных и графического интерфейса [4].

## 

## 4.3 Разработка кода

В первую очередь была проведена настройка сцены для дальнейшей разработки. Сюда относится подготовка движка к работе, подключение библиотек, плагинов и ресурсов, настройка сцены. Программный код представлен в приложении Б.

Для корректного отображения элементов на 3D сцене необходимо настроить следующие компоненты:

* ресурсы
* камера
* сцена
* покадровый рендеринг графики
* конфигурация (подгрузка плагинов и ресурсов)
* наблюдатели (Observed)

### 

### 4.3.1 Настройка подгружаемых ресурсов

Подгружаемые ресурсы передаются в Ogre из конфигурационного файла, в котором указаны пути к файлам, нужные в целях рендера. Как правило, это конфигурационный файл ресурсов, включающий в себя шрифты, 3D модели, файлы партиклов и так далее.

FileSystem=../../media/fonts

FileSystem=../../media/models

FileSystem=../../media/particle

А также конфигурационный файл плагинов, включающий в себя плагины графических библиотек DirectX, OpenGL, плагин партикловых эффектов и так далее.

Plugin=RenderSystem\_Direct3D9\_d

Plugin=RenderSystem\_GL\_d

Plugin=Plugin\_ParticleFX\_d

Таким образом, нужно указать приложению, какие файлы требуется подгружать на старте программы. Создаётся экземпляр класса Root, являющийся корневым классом и представляет собой отправную точку клиентского приложения, с помощью которого можно получить доступ к основам системы, а именно системам рендеринга, управлению конфигурациями, логированию и другим классам системы [10]. Также создаётся два экземпляра для работы с конфигурационными файлами плагинов и ресурсов, загружаемые при помощи функции load, которой передаются соответствующие названия файлов, загружает данные файлы.

После подгрузки ресурсов и плагинов требуется их проинициализировать, и задать начало работы рендера. Инициализация подразумевает указание направления передачи данных для подгруженного файла.

### 4.3.2 Покадровый рендеринг графики и шаблон наблюдатель

В начале создаётся функция FrameListener, отвечающая за прослушивание каждого кадра на экране рендера. FrameListener основан на паттерне наблюдателя (Observed). В функции создаётся список параметров для OIS (объектно-ориентированная система ввода), затем нужно получить от OGRE хэндл окна (дескриптор в виде числа, с помощью которого идентифицируется ресурс), в котором в текущий момент происходит рендеринг графики. Далее в систему ввода передаётся хэндл окна рендера и описываются функции обратного вызова (callback) для настройки считывания действий пользователя с мыши и клавиатуры. В завершение этой функции добавляется наблюдатель на текущее окно, на случай, если нужно обработать, например, изменение размера экрана рендера, то есть разворачивание окна.

Параллельно для рендера создаётся функция-слушатель frameRenderingQueued, которая принимает единственный параметр FrameEvent (кадр – единичное изображение экрана), таким образом, функция способна обрабатывать каждый кадр окна, поэтому здесь фиксируются нажатия клавиатуры и мыши.

Пример обработки клавиши ESC.

if (Keyboard->isKeyDown(OIS::KC\_ESCAPE))

return false;

### 4.3.3 Создание сцены, камеры и порта просмотра

Порт просмотра используется для отрисовки трёхмерного содержимого в 2D окне. Порт просмотра напрямую привязан к камере.

Для создания и дальнейшей настройки сцены, необходимо создать экземпляр класса менеджера сцены, способный управлять объектами, находящимися на сцене.

Прежде всего, необходимо настроить камеру для текущей сцены и задать параметры камеры, такие как позиция камеры в пространстве и направление камеры. Так как пространство на сцене трёхмерное, то для установки параметров камеры используются трёхмерные вектора, задающие три координаты в пространстве. Так как реализации камеры недостаточно для корректного отображения сцены на экране, необходимо создать порт просмотра, который является дочерним элементом камеры и с помощью него передать камере такие параметры как текущая ширина и высота экрана.

После настройки камеры для отображения объектов на сцене, необходимо настроить освещение сцены. В сцене используются два источника света: глобальное освещение и точечное освещение. Глобальное освещение используется для равномерного освещения сцены, а точечное в свою очередь для подсветки нужного места на сцене. Настройки освещения зависят от следующих параметров: цветовая гамма формата RGB, позиция источника света, направление освещения. Для настройки направления освещения в данном случае используется нормализация вектора.

### 4.3.4 Загрузка и установка заданных ресурсов

Отображение объекта на сцене требует явного описания его параметров в коде. Для того чтобы автоматизировать этот процесс, нужно знать какие 3D модели находятся в файле описания ресурсов и затем во время запуска приложения описать параметры всех 3D моделей. Таким образом после запуска все 3D модели появятся на сцене.

Для того чтобы узнать какие модели загружаются при запуске, создан вектор строкового типа, в который записываются названия подгружаемых 3D моделей. Так как все модели упаковываются в zip архив перед загрузкой на сцену, они программно отыскиваются по расширению .zip.

Средства движка сканируют конфигурационный файл построчно и записывают всю строку в отдельный указатель, который далее необходимо считать.

Поиск работает следующим образом: если в текущей строке есть подстрока «.zip», то с конца строки доходим до первого слеша (корневая директория файла) и далее считывается имя архива. Как правило имя архива должно совпадать с названием 3D модели. После завершения считывания имени оно заносится в созданный вектор.

char result[100];

if ( std::strstr(stringToChar, ".zip") )

{

int len = strlen(stringToChar);

int i = 0;

for(i = len; i > 0 ; i--)

{

if ( stringToChar[i-1] == '/')

{

break;

}

}

int res = 0;

for(int j = i; j < len ; j++)

{

result[res++] = stringToChar[j];

if ( stringToChar[j+1] == '.')

{

break;

}

}

myVector.push\_back( std::string(result) );

}

Далее требуется настроить параметры всех моделей, занесенных в вектор. Для того чтобы прикрепить 3D модель к сцене, Ogre3D использует узел сцены, являющийся дочерним элементов корневого узла сцены. Чтобы узел стал активным, нужно прикрепить к созданному узлу сцены сущность в виде 3D модели.

Так как требуется прикрепить сразу множество моделей на сцену, прикрепление реализовано в цикле. Проходим циклом по всему вектору, забираем из каждой позиции вектора название модели и создаём сущность с таким же именем и конкатенируем .mesh , так как в архиве файл 3D модели имеет именно такое расширение. Далее создаём узел сцены с именем модели, чтобы в дальнейшем была возможность обратиться к конкретного узлу по его имени и устанавливаем позицию узла на сцене.

for(int i = 0; i < myVector.size(); i++)

{

Ogre::Entity\* lEntity = mSceneMgr->createEntity( myVector[i] + ".mesh" );

lNode = mSceneMgr->getRootSceneNode()->createChildSceneNode(myVector[i]);

lNode->attachObject(lEntity);

lNode->setPosition(0, 0, 0);

}

### 4.3.5 Выделение моделей на сцене, управление камерой

Для взаимодействия пользователя с объектами на сцене, нужно реализовать обратную связь с пользователем. Взаимодействие со сценой происходит при помощи мыши, при нажатии на объект клавишей мыши, камера «подлетает» к выделенному объекту.

В целях реализации обратной связи необходимо определить нажатие пользователем на 3D модель на сцене. Для этого используется экземпляр класса Ray (луч), выпущенный из позиции нажатия мыши на экран. Также создаётся экземпляр очереди, связанный с лучом, в который записываются все объекты, через которые прошёл луч. В итоге, в цикле проходим по всей очереди и проверяем, является ли какой-либо из объектов, на который упал луч, одной из наших подгруженных моделей. Если объект является нашей моделью, то выделяем его. При возвращении камеры на исходную позицию снимаем выделение с 3D модели.

После настройки выбора модели на сцене, необходимо настроить камеру. Для настройки камеры нужно рассчитать траекторию от позиции камеры до позиции выделенной модели для каждого кадра, иначе говоря, требуется указать камере, в какую точку нужно двигаться каждый кадр. Для того чтобы рассчитать требуемую позицию для камеры на каждый кадр, нужно из позиции модели вычесть текущую позицию камеры и умножить на время, прошедшее с последнего кадра. Таким образом, когда разница координат модели и камеры будет равна нулю, камера достигнет нужной точки и остановится. Также во время движения камеры нужно указывать, на какую точку на сцене направлена камера, в нашем случае это координаты выделенной модели плюс небольшой отступ, иначе камера будет находиться в самой модели.

Возвращение камеры на исходную позицию происходит аналогичным образом. Исходная позиция камеры предустановлена в коде.

Ogre::Vector3 needPosition = SceneMgr

->getSceneNode(myVector[i])->getPosition() +

Ogre::Vector3(200,50,50);

Camera->move( (needPosition - Camera->getPosition() ) \* fe.timeSinceLastFrame);

Camera->lookAt(SceneMgr->getSceneNode(myVector[i]) ->getPosition() );

### 4.3.6 Сохранение состояния программы

В ходе разработке приложения выяснилось, что параметры моделей на сцене не сохраняются после закрытия приложения. После внедрения манипулирования моделями проблема сохранения стала актуальной. Для сохранения параметров моделей, а именно позиции модели, угла поворота и масштаба после закрытия программы реализовано сохранение в xml файле. Для работы с xml файлами подключена библиотека TinyXML.

Сохраняются следующие данные по всем осям: позиция модели в пространстве и масштаб модели, также сохраняется угол поворота модели.

После загрузки ресурсов и задания позиции каждой модели, в xml файл в качестве тегов записываются названия всех моделей, которые присутствуют в конфигурационном файле. Затем у каждого тега создаются атрибуты с параметрами модели, при этом если такая модель была загружена впервые, то параметры задаются по умолчанию. Также, чтобы не хранить лишнюю информацию в xml файле, при инициализации приложения происходит проверка, что в xml хранятся актуальные названия моделей, которые были загружены на старте приложения, и будут использоваться на сцене, иначе модель, не использующаяся на сцене, удаляется из xml файла с настройками.

if (rewrite)

{

xml\_name\_model->QueryDoubleAttribute("xPosition", &xPos);

xml\_name\_model->QueryDoubleAttribute("yPosition", &yPos);

xml\_name\_model->QueryDoubleAttribute("zPosition", &zPos);

locationVector.x = xPos;

locationVector.y = yPos;

locationVector.z = zPos;

xml\_name\_model->QueryDoubleAttribute("xScale", &xScale);

xml\_name\_model->QueryDoubleAttribute("yScale", &yScale);

xml\_name\_model->QueryDoubleAttribute("zScale", &zScale);

scaleVector.x = xScale;

scaleVector.y = yScale;

scaleVector.z = zScale;

xml\_name\_model->QueryDoubleAttribute("rotate", &rotateModel);

rotate = rotateModel;

}

if (addNew)

{

file\_settings->LinkEndChild(xml\_name\_model);

std::srand( std::time(NULL) );

locationVector.x = 0;

locationVector.y = 0;

locationVector.z = std::rand()%300;

xml\_name\_model->SetDoubleAttribute("xPosition", locationVector.x );

xml\_name\_model->SetDoubleAttribute("yPosition", ЯlocationVector.y );

xml\_name\_model->SetDoubleAttribute("zPosition", locationVector.z );

scaleVector.x = 1;

scaleVector.y = 1;

scaleVector.z = 1;

xml\_name\_model->SetDoubleAttribute("xScale", scaleVector.x);

xml\_name\_model->SetDoubleAttribute("yScale", scaleVector.y);

xml\_name\_model->SetDoubleAttribute("zScale", scaleVector.z);

rotate = 0.0;

xml\_name\_model->SetDoubleAttribute("rotate", rotate);

file\_settings->SaveFile();

}

Таким образом, в xml файле с параметрами модели всегда хранится актуальная информация о параметрах для каждой оси, кроме угла поворота, который не зависит от оси.

<exampleModel xPosition="19" yPosition="168" zPosition="77" xScale="0" yScale="0" zScale="0" rotate="-86" />

## 4.4 Разработка пользовательского интерфейса

GUI – graphical user interface (графический пользовательский интерфейс).

Интерфейс приложения реализован при помощи сторонней графической библиотеки CEGUI. В интерфейсе реализованы кнопки возврата камеры на исходную позицию, кнопка выхода, текстовые поля и интерфейс меню настроек.

### 

### 4.4.1 Интерфейс главного экрана

Каркасом графического интерфейса является менеджер виджетов к которому дочерними объектами прикрепляются все графические элементы. Исходные файлы всех графических элементов хранятся в полигональном атласе, который также подгружается при запуске программы. Параметры графических элементов в атласе описаны в xml файлах, поэтому при создании кнопки происходит обращение сначала к названию атласа, а затем к элементу этого атласа, например OgreTray/Button [8].

Для расположения GUI элементов на экране, используются UV-координаты – текстурные координаты, содержащие информацию для наложения текстуры.

Например, реализация кнопки выхода выглядит следующим образом: создаётся экземпляр кнопки, далее при помощи менеджера виджетов кнопке задаётся исходный текстурный файл из атласа. Также необходимо настроить текст кнопки, размер, расположение на экране и задать в обработчике кнопки к какой функции она будет обращаться при нажатии.

CEGUI::WindowManager &wmgr = CEGUI::WindowManager::getSingleton();

CEGUI::Window \*buttonQuit;

buttonQuit = wmgr.createWindow("OgreTray/Button", "CEGUI/QuitButton");

buttonQuit->setText(“Выход”);

buttonQuit->setSize(CEGUI::UVector2(CEGUI::UDim(0.15, 0), CEGUI::UDim(0.05, 0)));

buttonQuit->setPosition(CEGUI::UVector2(CEGUI::UDim(0, 0), CEGUI::UDim(0.05, 0)));

sheetMgr->addChildWindow(buttonQuit);

CEGUI::System::getSingleton().setGUISheet(sheetMgr);

buttonQuit->subscribeEvent(CEGUI::PushButton::EventClicked, CEGUI::Event::Subscriber(&BasicApp::quit, this));

Также необходимо добавить тестовое поле, которое будет выводить информацию о модели в том случае, если камера указывает на модель. Так как графическая библиотека не поддерживает кириллические символы по умолчанию, нужно реализовать конвертацию из кодировки юникод (Unicode) в кодировку широкого символа (WCHAR).

CEGUI::String encodeRussian(wchar\_t string[100])

{

wchar\_t strw[100] = L"";

wcscat(strw, string);

CEGUI::String str;

for (int i=0; i < wcslen(strw); i++)

{

str += strw[i];

}

return str;

}

Также, чтобы пользователь смог изменять и задавать текст для определённой модели, реализовано считывание текста с xml файла. При выделении объекта на сцене, запрашивается имя этого объекта, и в зависимости от него происходит обращение к конкретному файлу с описанием выбранного объекта. Текст считывается и конвертируется в нужную кодировку и выводится на экран. При возвращении камеры на исходную позицию текст скрывается.

### 

### 4.4.2 Меню настроек

Для удобного манипулирования 3D моделями реализовано меню настроек, которое включает в себя манипуляцию объектов при помощи пользовательского интерфейса. Манипулирование включает в себя: перемещение модели в пространстве, изменение угла поворота модели и изменение масштаба модели.

При зажатии, например, кнопки увеличения модели в интерфейсе GUI, каждый кадр происходит её одновременное масштабирование по всем осям в трёхмерном пространстве. По аналогии работают остальные кнопки манипулирования модели.

Реализация кода масштабирования модели выглядит так: в цикле проходим по списку моделей и с выделенной моделью из списка каждый кадр проводим масштабирование на 1.001 от предыдущего размера модели по всем осям.

for(int i = 0; i < myVector.size(); i++)

{

if ( mSceneMgr->getSceneNode(myVector[i])

->getShowBoundingBox() && static\_cast<CEGUI::TabButton\*>(scalePlusButton)->isPushed() )

{

Ogre::Vector3 needScale = mSceneMgr

->getSceneNode(myVector[i])->getScale() \* 1.001f;

mSceneMgr->getSceneNode(myVector[i])->setScale(needScale);

}

}

Учитывая то, что приложение не ограничено на количество подгружаемых пользователем 3D моделей, а также после реализации манипулирования с моделями логично добавить масштабирование комнаты в целях увеличение пространства на случай переизбытка моделей. Чтобы равномерно увеличить комнату, требуется масштабировать её только по двум осям.

if (static\_cast<CEGUI::TabButton\*>(scaleRoomPlusButton)->isPushed() )

{

Ogre::Real needScaleX = mSceneMgr

->getSceneNode(“Room”)->getScale().x \* 1.001f;

Ogre::Real needScaleY = mSceneMgr

->getSceneNode(“Room”)->getScale().y;

Ogre::Real needScaleZ = mSceneMgr

->getSceneNode(“Room”)->getScale().z \* 1.001f;

mSceneMgr->getSceneNode(“Room”)->setScale(needScaleX, needScaleY, needScaleZ);

}

## 4.5 Тестирование

Тестирование и отладка является важным этапом разработки программного продукта. Чтобы оператор смог пользоваться приложением без возникновения исключений и ошибок, требуется отладить программный код. В целях выявления ошибок в приложении было проведено модульное тестирование, которое позволяет проверить на корректность все модули программного кода [9].

В ходе работы было проведено тестирование программного продукта в целях обнаружения и исправления ошибок в коде.

Прежде чем приступить к разработке интерфейса, и подключения сторонних библиотек было проведено тестирование основного каркаса приложения, в которое входит выбор моделей и загрузка ресурсов, чтобы исключить выделение нескольких моделей на сцене.

Затем было проведено тестирование пользовательского интерфейса и загрузки нужных ресурсов. В целях сокращения нежелательных действий пользователя при использовании интерфейса, многие кнопки блокируются при выполнении некоторых действий. Также при работе с xml файлом с настройками протестирована корректная работа файла при загрузке сразу нескольких моделей, а также очистка файла, когда в нём содержится информация о не загружаемых файлах при запуске приложения.

Чтобы убедиться, что программный продукт запускается и работает корректно на различных процессорах, был проведён запуск приложения на различных рабочих станциях в целях выявления результатов производительности приложения. Также после завершения разработки интерфейса проведена проверка эргономичности продукта, чтобы убедится в удобстве пользовательского интерфейса.

# 5. Разработка руководства пользователя

В целях предоставления пользователю информации по работе с программой разработано руководство пользователя, включающее в себя:

* инструкцию по разработке 3D моделей и дальнейшему их экспортированию для использования в приложении
* инструкцию по использованию 3D моделей в программе и настройку их параметров через xml файл
* инструкцию по общему использованию приложения

Руководство пользователя представлено в приложении А.

# Заключение

В дипломной работе решены следующие задачи:

1. Исследованы существующие системы визуализации
2. Исследованы подходящие библиотеки для разработки 3D сцены
3. Возможность загрузки и отображения множества объектов на сцене
4. Реализована и протестирована система визуализации

Разработанную систему визуализации можно использовать для предоставления детальной и подробной информации целевой аудитории.

# Список используемой литературы

1. Ogre 3D [Электронный ресурс]: офиц. сайт. – Режим доступа: http://www.ogre3d.org
2. OpenSceneGraph [Электронный ресурс]: офиц. сайт. – Режим доступа: http://www.openscenegraph.org
3. UnrealEngine [Электронный ресурс]: офиц. сайт. – Режим доступа: https://www.unrealengine.com
4. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссиде. — СПб: Питер, 2001. — 368 с.
5. 3D тур по горному университету [Электронный ресурс]: офиц. сайт. – Режим доступа: http://3dtour.spmi.ru
6. Boeing 777-200ER – 3D панорама (Виртуальный тур) [Электронный ресурс]: офиц. сайт. – Режим доступа: http://www.orenair.ru/userfiles/panorama/Boeing-777/rus/tour.html
7. 3D путешествие по отелю Галакт [Электронный ресурс]: офиц. сайт. – Режим доступа: http://www.galakt.ru/3d-puteshestvie-po-otelyu-galakt/
8. CEGUI [Электронный ресурс]: офиц. сайт. – Режим доступа: http://cegui.org
9. Бейзер, Б. Тестирование черного ящика. Технологии функционального тестирования программного обеспечения и систем / Б. Бейзер. – СПб: Питер, 2004. – 320 с.
10. Kerger, F. Ogre 3D 1.7 Beginner’s Guide / F. Kerger – Birmingham: Packt Publishing Ltd, 2010. – 300 p.

# 

# Приложения

# Приложение А

Руководство пользователя

Программа для запуска приложения расположена в папке Demo и называется Present3D.exe. Для корректной работы программы, во время её старта из выпадающего списка нужно выбрать графическую библиотеку для рендера DirectX. В случае если на машине не установлена данная библиотека, в папке Demo приложен файл для установки dxwebsetup.exe.

Разработка и загрузка собственных 3D моделей.

Для разработки моделей потребуется Blender 2.66. После разработки анимации у 3D модели, обязательно нужно назвать анимацию Walk.

После разработки 3D моделей, требуется настроить экспортер для использования моделей в движке. В папке с приложением находится файл io\_export\_ogreDotScene.py, который нужно переместить в расположение *C:\Program Files\Blender\2.66\scripts\addons*. После перемещения нужно зайти в Blender и активировать данный скрипт, для этого выполнить действия:

* Перейти в User Preferences
* Открыть вкладку Addons
* Найти в списке OGRE Exporter и активировать галочку на этом скрипте

Экспорт модели проводится в меню File > Export > Ogre3D (.scene and .mesh), затем нужно выбрать директорию для сохранения файла.

Следующие файлы создаются при экспорте модели из Blender:

* .mesh – основной файл модели
* .material – файл материала (если материал применялся)
* .skeleton – файл анимации (если модель имеет анимацию)

Все такие файлы после экспорта требуется упаковать в zip архив и переместить в папку с приложением по пути *Demo/media/packs.*

После перемещения архива с моделью требуется добавить в конфигурационный файл *resources.cfg* путь к созданной модели в таком виде *Zip=media/packs/NameModel.zip*

**На месте NameModel должно быть такое имя модели, которое содержит расширение .mesh в архиве.**

Для добавления описания в виде текста к модели, нужно создать текстовый файл с названием, **которое содержит расширение .mesh в архиве,** и поместить в расположение *Demo\media\text.*

Работа с программой.

При запуске программы перед пользователем появляются созданные им модели. При выборе модели мышкой, камера подлетает к модели и затем появляется информационный текст о модели, а также анимация модели, если она существует. Для того, чтобы вернуться на исходную позицию камеры, нужно нажать кнопку «Вернуться». Кнопка «Остановить камеру» включает режим свободного манипулирования камерой при помощью мыши и кнопок WASD.

Меню настроек.

В меню настроек можно манипулировать моделями, для этого требуется перейти в меню настроек, выделить мышкой модель и с помощью кнопок манипулирования можно перемещать модель в пространстве и разворачивать вокруг своей оси.

# Приложение Б

Программный код

BasicApp.h

#ifndef BASICAPP\_H

#define BASICAPP\_H

#include <OgreRoot.h>

#include <OgreCamera.h>

#include <OgreViewport.h>

#include <OgreSceneManager.h>

#include <OgreRenderWindow.h>

#include <OgreConfigFile.h>

#include <OgreException.h>

#include <OgreEntity.h>

#include <OgreFrameListener.h>

#include <OgreWindowEventUtilities.h>

#include <Terrain\OgreTerrain.h>

#include <Terrain\OgreTerrainGroup.h>

#include <OgreSceneQuery.h>

#include <OISEvents.h>

#include <OISInputManager.h>

#include <OISKeyboard.h>

#include <OISMouse.h>

#include <CEGUI.h>

#include <RendererModules/Ogre/CEGUIOgreRenderer.h>

#include <SdkCameraMan.h>

#include <cstring>

#include <vector>

#include <Windows.h>

#include "textMgr.h"

#include <tinyxml.h>

#include <algorithm>

#include <stdlib.h>

class BasicApp

: public Ogre::WindowEventListener,

public Ogre::FrameListener,

public OIS::KeyListener,

public OIS::MouseListener

{

public:

BasicApp();

~BasicApp();

void go();

private:

bool mShutdown;

//Setup - Main

Ogre::Root\* mRoot;

Ogre::Camera\* mCamera;

Ogre::SceneManager\* mSceneMgr;

Ogre::RenderWindow\* mWindow;

Ogre::String mResourcesCfg;

Ogre::String mPluginsCfg;

OgreBites::SdkCameraMan\* mCameraMan;

Ogre::SceneNode\* lNode;//записываем подгружаемые меши

Ogre::Entity\* room;

Ogre::SceneNode\* roomNode;

Ogre::AnimationState\* anim;

//Setup - Render

virtual bool frameRenderingQueued(const Ogre::FrameEvent& fe);

//Setup - Keyboard

virtual bool keyPressed(const OIS::KeyEvent& ke);

virtual bool keyReleased(const OIS::KeyEvent& ke);

float mRotSpd;

bool mMovableFound;

bool mLMouseDown, mRMouseDown;

Ogre::Vector3 prevPositionModel;

//Setup - Mouse

virtual bool mouseMoved(const OIS::MouseEvent& me);

virtual bool mousePressed(const OIS::MouseEvent& me, OIS::MouseButtonID id);

virtual bool mouseReleased(const OIS::MouseEvent& me, OIS::MouseButtonID id);

//Setup - Window

virtual void windowResized(Ogre::RenderWindow\* rw);

virtual void windowClosed(Ogre::RenderWindow\* rw);

//Setup - application

bool setup();

bool configure();

void chooseSceneManager();

void createCamera();

void createScene();

void destroyScene();

void createFrameListener();

void createViewports();

void setupResources();

void createResourceListener();

void loadResources();

// CEGUI

CEGUI::OgreRenderer\* mRenderer;

bool setupCEGUI();

// OIS

OIS::Mouse\* mMouse;

OIS::Keyboard\* mKeyboard;

OIS::InputManager\* mInputMgr;

//Terrain

Ogre::TerrainGroup\* mTerrainGroup;

Ogre::TerrainGlobalOptions\* mTerrainGlobals;

//Setup - Scene

Ogre::RaySceneQuery\* mRayScnQuery;//луч, опрашиваемый сцену

bool goDefaultPosition;//останавливает камеру при нажатии

void handleCameraCollision();

//GUI - Buttons

CEGUI::Window \*buttonCameraSetPosition;//возвращения камеры на позицию

CEGUI::Window \*buttonQuit;//выход

CEGUI::Window\* textbox1;//текст для модели

CEGUI::Window\* textShadow;

CEGUI::Window\* menuButton; //меню настроек

CEGUI::Window\* scalePlusButton; //увеличение скейла

CEGUI::Window\* scaleMinusButton; //уменьшение скейла

CEGUI::Window\* translateModelToLeftButton; //модельку влево

CEGUI::Window\* translateModelToRightButton; //модельку вправо

CEGUI::Window\* translateModelToTopButton; //модельку вперёд

CEGUI::Window\* translateModelToBackButton; //модельку назад

CEGUI::Window\* translateModelToUpButton; //модельку вверх

CEGUI::Window\* translateModelToDownButton; //модельку вниз

CEGUI::Window\* rotateRightModelButton; //поворачиваем модельку вправо

CEGUI::Window\* rotateLeftModelButton; //поворачиваем модельку влево

CEGUI::Window\* backMainWindowButton;//назад из меню настроек на главную

CEGUI::Window\* checkShadow;

CEGUI::Window\* releaseCamera;

CEGUI::Window\* tipsText;

CEGUI::Window\* tipsSettingText;

CEGUI::Window\* tipsRoomText;

CEGUI::Window\* tipsObjectText;

CEGUI::Window\* scaleRoomPlusButton; //увеличение скейла

CEGUI::Window\* scaleRoomMinusButton; //уменьшение скейла

//GUI - Subscribes buttons

void eventRoomScalePlus();

void eventRoomScaleMinus();

void eventModelScalePlus();

void eventModelScaleMinus();

void eventModelTranslateLeft();

void eventModelTranslateRight();

void eventModelTranslateTop();

void eventModelTranslateBack();

void eventModelTranslateUp();

void eventModelTranslateDown();

void eventModelRotateRight();

void eventModelRotateLeft();

bool eventMenuButton(const CEGUI::EventArgs &e);

bool quit(const CEGUI::EventArgs &e);//обработчик кнопки выхода

bool setDefaultPosition(const CEGUI::EventArgs &e);//обработчик кнопки cameraDefaultPosition

bool eventBackWindow(const CEGUI::EventArgs &e);//обработчик кнопки backMainWindow

void checkShadows();

//Timers

Ogre::Timer timer1;

Ogre::Timer timer2;

//Work with text model

std::vector<std::string> myVector; //вектор названий мешей в конфиге

char result[100]; //выходное название меша

CEGUI::String encodeRussian(wchar\_t string[100]); //декодируем в русский текст

//Inside classes & SDK

textMgr\* textManager;

TiXmlDocument \*file\_settings;

TiXmlDocument \*file\_room\_settings;

//GUI - Alpha

void goInAlpha(CEGUI::Window\* win); //функция ухода в альфу

void goFromAlpha(CEGUI::Window\* win); //функция выхода из альфы

bool goFromAlphaButton;//если моделька выделена, вытаскиваем кнопку из альфы

//GUI - Widgets

CEGUI::Window\* sheetMgr;//менеджер виджетов

CEGUI::Window \*sheetMainWindow;//виджет главного окна

CEGUI::Window \*sheetMainMenu;//виджет меню настроек

//Setup - XML setting

void configureXmlSettings();

void removeFromXmlElderData();

void configureXmlSettingsD(const char \* nameModel, Ogre::Vector3 &locationVector, Ogre::Vector3 &scaleVector, double &rotate, Ogre::Vector3 &scaleRoomVector);

bool tipsEnabled;

};

#endif

BasicApp.cpp

#include "BasicApp.h"

BasicApp::BasicApp()

: mShutdown(false),

mRoot(0),

mCamera(0),

mSceneMgr(0),

mWindow(0),

mResourcesCfg(Ogre::StringUtil::BLANK),

mPluginsCfg(Ogre::StringUtil::BLANK),

mCameraMan(0),

mRenderer(0),

mMouse(0),

mKeyboard(0),

mInputMgr(0),

mTerrainGroup(0),

mTerrainGlobals(0),

mRotSpd(0.1),

mLMouseDown(false),

mRMouseDown(false),

mRayScnQuery(0),

goDefaultPosition(false),

goFromAlphaButton(false),

sheetMainWindow(0),

textManager(0),

sheetMainMenu(0),

sheetMgr(0)

{

myVector.clear();

file\_settings = new TiXmlDocument("modelSettings.xml");

file\_room\_settings = new TiXmlDocument("settings.xml");

}

BasicApp::~BasicApp()

{

if (mCameraMan) delete mCameraMan;

mSceneMgr->destroyQuery(mRayScnQuery);

Ogre::WindowEventUtilities::removeWindowEventListener(mWindow, this);

windowClosed(mWindow);

delete mRoot;

}

void BasicApp::go()

{

#ifdef \_DEBUG

mResourcesCfg = "resources\_d.cfg";

mPluginsCfg = "plugins\_d.cfg";

#else

mResourcesCfg = "resources.cfg";

mPluginsCfg = "plugins.cfg";

#endif

if (!setup())

return;

mRoot->startRendering();

destroyScene();

}

bool BasicApp::frameRenderingQueued(const Ogre::FrameEvent& fe)

{

if (mKeyboard->isKeyDown(OIS::KC\_ESCAPE))

mShutdown = true;

if (mShutdown)

return false;

if (mWindow->isClosed())

return false;

mKeyboard->capture();

mMouse->capture();

mCameraMan->frameRenderingQueued(fe);

eventRoomScalePlus();

eventRoomScaleMinus();

eventModelScalePlus();

eventModelScaleMinus();

eventModelTranslateLeft();

eventModelTranslateRight();

eventModelTranslateTop();

eventModelTranslateBack();

eventModelTranslateUp();

eventModelTranslateDown();

eventModelRotateRight();

eventModelRotateLeft();

checkShadows();

for(int i = 0; i < myVector.size(); i++)

{

//если активно главное окно и выделена моделька

if ( mSceneMgr->getSceneNode(myVector[i])->getShowBoundingBox() && sheetMainWindow->isActive() )

{

Ogre::Vector3 needPosition = mSceneMgr->getSceneNode(myVector[i])->getPosition() + Ogre::Vector3(200,50,50);

mCamera->move( (needPosition - mCamera->getPosition() ) \* fe.timeSinceLastFrame );

mCamera->lookAt(mSceneMgr->getSceneNode(myVector[i])->getPosition() + Ogre::Vector3(0,50,50) );

textbox1->setVisible(true);

prevPositionModel = mSceneMgr->getSceneNode(myVector[i])->getPosition();

tipsEnabled = false;

wchar\_t cameraText[100] = L"Вернуться";

buttonCameraSetPosition->setText( encodeRussian(cameraText) );

if (mSceneMgr->getEntity(myVector[i])->hasAnimationState("Walk"))

{

anim = mSceneMgr->getEntity(myVector[i])->getAnimationState("Walk");

anim->setLoop(true);

anim->setEnabled(true);

anim->addTime(fe.timeSinceLastFrame);

}

}

}

if (goDefaultPosition)

{

Ogre::Vector3 needPosition(500, 350, 150);

mCamera->move( (needPosition - mCamera->getPosition() ) \* fe.timeSinceLastFrame );

mCamera->lookAt(prevPositionModel);

wchar\_t cameraText[100] = L"Остановить камеру";

buttonCameraSetPosition->setText( encodeRussian(cameraText) );

for(int i = 0; i < myVector.size(); i++)

{

if ( mSceneMgr->getSceneNode(myVector[i])->getShowBoundingBox() && mSceneMgr->getEntity(myVector[i])->hasAnimationState("Walk"))

{

anim->setEnabled(false);

}

}

}

tipsText->setVisible(tipsEnabled);

CEGUI::System::getSingleton().injectTimePulse(fe.timeSinceLastFrame);

return true;

}

void BasicApp::eventRoomScalePlus()

{

if (static\_cast<CEGUI::TabButton\*>(scaleRoomPlusButton)->isPushed() )

{

Ogre::Real needScaleX = roomNode->getScale().x \* 1.001f;

Ogre::Real needScaleY = roomNode->getScale().y;

Ogre::Real needScaleZ = roomNode->getScale().z \* 1.001f;

roomNode->setScale(needScaleX, needScaleY, needScaleZ);

}

}

void BasicApp::eventRoomScaleMinus()

{

if (static\_cast<CEGUI::TabButton\*>(scaleRoomMinusButton)->isPushed() )

{

Ogre::Real needScaleX = roomNode->getScale().x \* 0.999f;

Ogre::Real needScaleY = roomNode->getScale().y;

Ogre::Real needScaleZ = roomNode->getScale().z \* 0.999f;

roomNode->setScale(needScaleX, needScaleY, needScaleZ);

}

}

void BasicApp::eventModelScalePlus()

{

for(int i = 0; i < myVector.size(); i++)

{

if ( mSceneMgr->getSceneNode(myVector[i])->getShowBoundingBox() && static\_cast<CEGUI::TabButton\*>(scalePlusButton)->isPushed() )

{

Ogre::Vector3 needScale = mSceneMgr->getSceneNode(myVector[i])->getScale() \* 1.001f; //костыль на скейл

mSceneMgr->getSceneNode(myVector[i])->setScale(needScale);

}

}

}

void BasicApp::eventModelScaleMinus()

{

for(int i = 0; i < myVector.size(); i++)

{

if ( mSceneMgr->getSceneNode(myVector[i])->getShowBoundingBox() && static\_cast<CEGUI::TabButton\*>(scaleMinusButton)->isPushed() )

{

Ogre::Vector3 needScale = mSceneMgr->getSceneNode(myVector[i])->getScale() \* 0.999f; //костыль на скейл

mSceneMgr->getSceneNode(myVector[i])->setScale(needScale);

}

}

}

void BasicApp::eventModelTranslateLeft()

{

for(int i = 0; i < myVector.size(); i++)

{

if ( mSceneMgr->getSceneNode(myVector[i])->getShowBoundingBox() && static\_cast<CEGUI::TabButton\*>(translateModelToLeftButton)->isPushed() )

{

mSceneMgr->getSceneNode(myVector[i])->translate(0.0f, 0.0f, 0.2f);

}

}

}

void BasicApp::eventModelTranslateRight()

{

for(int i = 0; i < myVector.size(); i++)

{

if ( mSceneMgr->getSceneNode(myVector[i])->getShowBoundingBox() && static\_cast<CEGUI::TabButton\*>(translateModelToRightButton)->isPushed() )

{

mSceneMgr->getSceneNode(myVector[i])->translate(0.0f, 0.0f, -0.2f);

}

}

}

void BasicApp::eventModelTranslateTop()

{

for(int i = 0; i < myVector.size(); i++)

{

if ( mSceneMgr->getSceneNode(myVector[i])->getShowBoundingBox() && static\_cast<CEGUI::TabButton\*>(translateModelToTopButton)->isPushed() )

{

mSceneMgr->getSceneNode(myVector[i])->translate(-0.2f, 0.0f, 0.0f);

}

}

}

void BasicApp::eventModelTranslateBack()

{

for(int i = 0; i < myVector.size(); i++)

{

if ( mSceneMgr->getSceneNode(myVector[i])->getShowBoundingBox() && static\_cast<CEGUI::TabButton\*>(translateModelToBackButton)->isPushed() )

{

mSceneMgr->getSceneNode(myVector[i])->translate(0.2f, 0.0f, 0.0f);

}

}

}

void BasicApp::eventModelTranslateUp()

{

for(int i = 0; i < myVector.size(); i++)

{

if ( mSceneMgr->getSceneNode(myVector[i])->getShowBoundingBox() && static\_cast<CEGUI::TabButton\*>(translateModelToUpButton)->isPushed() )

{

mSceneMgr->getSceneNode(myVector[i])->translate(0.0f, 0.2f, 0.0f);

}

}

}

void BasicApp::eventModelTranslateDown()

{

for(int i = 0; i < myVector.size(); i++)

{

if ( mSceneMgr->getSceneNode(myVector[i])->getShowBoundingBox() && static\_cast<CEGUI::TabButton\*>(translateModelToDownButton)->isPushed() )

{

mSceneMgr->getSceneNode(myVector[i])->translate(0.0f, -0.2f, 0.0f);

}

}

}

void BasicApp::eventModelRotateRight()

{

for(int i = 0; i < myVector.size(); i++)

{

if ( mSceneMgr->getSceneNode(myVector[i])->getShowBoundingBox() && static\_cast<CEGUI::TabButton\*>(rotateRightModelButton)->isPushed() )

{

mSceneMgr->getSceneNode(myVector[i])->yaw( Ogre::Degree(0.2f) );

}

}

}

void BasicApp::eventModelRotateLeft()

{

for(int i = 0; i < myVector.size(); i++)

{

if ( mSceneMgr->getSceneNode(myVector[i])->getShowBoundingBox() && static\_cast<CEGUI::TabButton\*>(rotateLeftModelButton)->isPushed() )

{

mSceneMgr->getSceneNode(myVector[i])->yaw( Ogre::Degree(-0.2f) );

}

}

}

void BasicApp::goInAlpha(CEGUI::Window\* win)

{

win->setAlpha( 100.0f / (timer1.getMilliseconds() ) ); //уход в альфу

}

void BasicApp::goFromAlpha(CEGUI::Window\* win)

{

win->setAlpha( 0.001f \* (timer1.getMilliseconds() ) ); //закинуть в функцию выход из альфы

}

bool BasicApp::keyPressed(const OIS::KeyEvent& ke)

{

CEGUI::System::getSingleton().injectKeyDown((CEGUI::Key::Scan)ke.key);

CEGUI::System::getSingleton().injectChar((CEGUI::Key::Scan)ke.text);

mCameraMan->injectKeyDown(ke);

return true;

}

bool BasicApp::keyReleased(const OIS::KeyEvent& ke)

{

CEGUI::System::getSingleton().injectKeyUp((CEGUI::Key::Scan)ke.key);

mCameraMan->injectKeyUp(ke);

return true;

}

bool BasicApp::mouseMoved(const OIS::MouseEvent& me)

{

CEGUI::System::getSingleton().injectMouseMove(me.state.X.rel, me.state.Y.rel);

if (mLMouseDown)

{

}

else if (mRMouseDown)

{

mCamera->yaw(Ogre::Degree(-me.state.X.rel \* mRotSpd));

mCamera->pitch(Ogre::Degree(-me.state.Y.rel \* mRotSpd));

}

return true;

}

// Helper function for mouse events

CEGUI::MouseButton convertButton(OIS::MouseButtonID id)

{

switch (id)

{

case OIS::MB\_Left:

return CEGUI::LeftButton;

case OIS::MB\_Right:

return CEGUI::RightButton;

case OIS::MB\_Middle:

return CEGUI::MiddleButton;

default:

return CEGUI::LeftButton;

}

}

bool BasicApp::mousePressed(const OIS::MouseEvent& me, OIS::MouseButtonID id)

{

CEGUI::System::getSingleton().injectMouseButtonDown(convertButton(id));

if (id == OIS::MB\_Left)

{

CEGUI::Point mousePos = CEGUI::MouseCursor::getSingleton().getPosition();

Ogre::Ray mouseRay =

mCamera->getCameraToViewportRay(

mousePos.d\_x / float(me.state.width),

mousePos.d\_y / float(me.state.height));

mRayScnQuery->setRay(mouseRay);

mRayScnQuery->setSortByDistance(true);

Ogre::RaySceneQueryResult& result = mRayScnQuery->execute();

Ogre::RaySceneQueryResult::iterator it = result.begin();

mMovableFound = false;

for ( ; it != result.end(); it++)

{

mMovableFound =

it->movable &&

it->movable->getName() != "" &&

it->movable->getName() != "PlayerCam";

for(int i = 0; i < myVector.size(); i++)

{

if (it->movable->getParentSceneNode()->getName() == myVector[i])

{

it->movable->getParentSceneNode()->showBoundingBox(true);

goDefaultPosition = false;

textbox1->setVisible(true);

textbox1->setText( encodeRussian (textManager->getFromFile( myVector[i]) ) );

timer1.reset(); //сбрасываем таймер для нулевой альфы

goFromAlphaButton = true;

break;

}

}

}

mLMouseDown = true;

}

else if (id == OIS::MB\_Right)

{

CEGUI::MouseCursor::getSingleton().hide();

mRMouseDown = true;

}

return true;

}

bool BasicApp::mouseReleased(const OIS::MouseEvent& me, OIS::MouseButtonID id)

{

CEGUI::System::getSingleton().injectMouseButtonUp(convertButton(id));

if (id == OIS::MB\_Left)

{

mLMouseDown = false;

}

else if (id == OIS::MB\_Right)

{

CEGUI::MouseCursor::getSingleton().show();

mRMouseDown = false;

}

return true;

}

void BasicApp::windowResized(Ogre::RenderWindow\* rw) //not used

{

unsigned int width, height, depth;

int left, top;

rw->getMetrics(width, height, depth, left, top);

const OIS::MouseState& ms = mMouse->getMouseState();

ms.width = width;

ms.height = height;

}

void BasicApp::windowClosed(Ogre::RenderWindow\* rw)

{

if (rw == mWindow)

{

if (mInputMgr)

{

mInputMgr->destroyInputObject(mMouse);

mInputMgr->destroyInputObject(mKeyboard);

OIS::InputManager::destroyInputSystem(mInputMgr);

mInputMgr = 0;

}

}

}

bool BasicApp::setup()

{

mRoot = new Ogre::Root(mPluginsCfg);

setupResources();

if (!configure())

return false;

chooseSceneManager();

createCamera();

createViewports();

Ogre::TextureManager::getSingleton().setDefaultNumMipmaps(5);

createResourceListener();

loadResources();

setupCEGUI();

createScene();

createFrameListener();

mRayScnQuery = mSceneMgr->createRayQuery(Ogre::Ray());

return true;

}

bool BasicApp::configure()

{

if (!(mRoot->restoreConfig() || mRoot->showConfigDialog()))

{

return false;

}

mWindow = mRoot->initialise(true, "Present3D");

return true;

}

void BasicApp::chooseSceneManager()

{

mSceneMgr = mRoot->createSceneManager(Ogre::ST\_EXTERIOR\_CLOSE);

}

void BasicApp::createCamera()

{

mCamera = mSceneMgr->createCamera("DefaultCam");

mCamera->setPosition(Ogre::Vector3(500, 350, 150));

mCamera->lookAt(Ogre::Vector3(0, 0, 150));

mCamera->setNearClipDistance(0.1);

mCamera->setFarClipDistance(50000);

mCameraMan = new OgreBites::SdkCameraMan(mCamera);

}

void BasicApp::createScene()

{

mSceneMgr->setAmbientLight(Ogre::ColourValue(0.7, 0.7, 0.7));

Ogre::Vector3 lightDir(0.55, 0.3, 0.75);

lightDir.normalise();

Ogre::Light\* light = mSceneMgr->createLight("SceneLight");

// light->setType(Ogre::Light::LT\_DIRECTIONAL);

light->setType(Ogre::Light::LT\_POINT);

light->setDirection(lightDir);

light->setDiffuseColour(Ogre::ColourValue(0.4, 0.4, 0.4));

light->setSpecularColour(Ogre::ColourValue(0.2, 0.2, 0.2));

light->setPosition(500,350,250);

//--------виджет главного окна + менеджер

CEGUI::WindowManager &wmgr = CEGUI::WindowManager::getSingleton();

sheetMgr = wmgr.createWindow("DefaultWindow", "sheetMgr");

sheetMainWindow = wmgr.createWindow("DefaultWindow", "sheetMainWindow");

sheetMainMenu = wmgr.createWindow("DefaultWindow", "sheetMainMenu");

//--------кнопка выхода

buttonQuit = wmgr.createWindow("OgreTray/Button", "CEGUIDemo/QuitButton");

wchar\_t quitText[100] = L"Выход";

buttonQuit->setText( encodeRussian(quitText) );

buttonQuit->setSize(CEGUI::UVector2(CEGUI::UDim(0.15, 0), CEGUI::UDim(0.05, 0)));

//--------кнопка возвращения камеры на исходную позицию

buttonCameraSetPosition = wmgr.createWindow("OgreTray/Button", "CEGUIDemo/buttonCameraSetPosition");

wchar\_t cameraText[100] = L"Выберите модель";

buttonCameraSetPosition->setText( encodeRussian(cameraText) );

buttonCameraSetPosition->setSize(CEGUI::UVector2(CEGUI::UDim(0.15, 0), CEGUI::UDim(0.05, 0)));

buttonCameraSetPosition->setPosition(CEGUI::UVector2(CEGUI::UDim(0, 0), CEGUI::UDim(0.05, 0)));

//--------текст модели

textbox1 = wmgr.createWindow("OgreTray/StaticText", "CEGUIDemo/TextBox1");

textbox1->setVisible(false);

textbox1->setSize(CEGUI::UVector2(CEGUI::UDim(0.35, 0), CEGUI::UDim(0.3, 0)));

textbox1->setPosition(CEGUI::UVector2(CEGUI::UDim(0.2, 0), CEGUI::UDim(0.3, 0)));

//--------текст тени

textShadow = wmgr.createWindow("OgreTray/ListboxItem", "CEGUIDemo/shadowText");

wchar\_t shadowText[100] = L"Тени";

textShadow->setText( encodeRussian(shadowText) );

textShadow->setSize(CEGUI::UVector2(CEGUI::UDim(0.1, 0), CEGUI::UDim(0.05, 0)));

textShadow->setPosition(CEGUI::UVector2(CEGUI::UDim(0, 0), CEGUI::UDim(0, 0)));

checkShadow = wmgr.createWindow("Vanilla/Checkbox", "CEGUIDemo/check");

checkShadow->setSize(CEGUI::UVector2(CEGUI::UDim(0.05, 0), CEGUI::UDim(0.05, 0)));

checkShadow->setPosition(CEGUI::UVector2(CEGUI::UDim(0.12, 0), CEGUI::UDim(0, 0)));

//главное меню

menuButton = wmgr.createWindow("OgreTray/Button", "menuButton");

wchar\_t settingsText[100] = L"Настройки";

menuButton->setText( encodeRussian(settingsText) );

menuButton->setSize(CEGUI::UVector2(CEGUI::UDim(0.15, 0), CEGUI::UDim(0.05, 0)));

menuButton->setPosition(CEGUI::UVector2(CEGUI::UDim(0.85, 0), CEGUI::UDim(0, 0)));

//Кнопка увеличения скейла

scalePlusButton = wmgr.createWindow("but/scalePlusButton", "CEGUIDemo/scalePlusButton");

scalePlusButton->setSize(CEGUI::UVector2(CEGUI::UDim(0.07, 0), CEGUI::UDim(0.07, 0)));

scalePlusButton->setPosition(CEGUI::UVector2(CEGUI::UDim(0.8, 0), CEGUI::UDim(0.6, 0)));

//Кнопка уменьшения скейла

scaleMinusButton = wmgr.createWindow("but/scaleMinusButton", "CEGUIDemo/scaleMinusButton");

scaleMinusButton->setSize(CEGUI::UVector2(CEGUI::UDim(0.07, 0), CEGUI::UDim(0.07, 0)));

scaleMinusButton->setPosition(CEGUI::UVector2(CEGUI::UDim(0.8, 0), CEGUI::UDim(0.7, 0)));

//Кнопка перемещения модели влево

translateModelToLeftButton = wmgr.createWindow("but/moveButton", "CEGUIDemo/translateModelToLeftButton");

translateModelToLeftButton->setRotation( CEGUI::Vector3(0,0,180) );

translateModelToLeftButton->setSize(CEGUI::UVector2(CEGUI::UDim(0.07, 0), CEGUI::UDim(0.07, 0)));

translateModelToLeftButton->setPosition(CEGUI::UVector2(CEGUI::UDim(0.58, 0), CEGUI::UDim(0.7, 0)));

//Кнопка перемещения модели вправо

translateModelToRightButton = wmgr.createWindow("but/moveButton", "CEGUIDemo/translateModelToRightButton");

translateModelToRightButton->setSize(CEGUI::UVector2(CEGUI::UDim(0.07, 0), CEGUI::UDim(0.07, 0)));

translateModelToRightButton->setPosition(CEGUI::UVector2(CEGUI::UDim(0.72, 0), CEGUI::UDim(0.7, 0)));

//Кнопка перемещения модели вперед

translateModelToTopButton = wmgr.createWindow("but/moveButton", "CEGUIDemo/translateModelToTopButton");

translateModelToTopButton->setRotation( CEGUI::Vector3(0,0,-90) );

translateModelToTopButton->setSize(CEGUI::UVector2(CEGUI::UDim(0.07, 0), CEGUI::UDim(0.07, 0)));

translateModelToTopButton->setPosition(CEGUI::UVector2(CEGUI::UDim(0.65, 0), CEGUI::UDim(0.6, 0)));

//Кнопка перемещения модели назад

translateModelToBackButton = wmgr.createWindow("but/moveButton", "CEGUIDemo/translateModelToBackButton");

translateModelToBackButton->setRotation( CEGUI::Vector3(0,0,90) );

translateModelToBackButton->setSize(CEGUI::UVector2(CEGUI::UDim(0.07, 0), CEGUI::UDim(0.07, 0)));

translateModelToBackButton->setPosition(CEGUI::UVector2(CEGUI::UDim(0.65, 0), CEGUI::UDim(0.71, 0)));

//Кнопка перемещения модели вверх

translateModelToUpButton = wmgr.createWindow("but/topButton", "CEGUIDemo/translateModelToUpButton");

translateModelToUpButton->setSize(CEGUI::UVector2(CEGUI::UDim(0.07, 0), CEGUI::UDim(0.07, 0)));

translateModelToUpButton->setPosition(CEGUI::UVector2(CEGUI::UDim(0.6, 0), CEGUI::UDim(0.64, 0)));

//Кнопка перемещения модели вверх

translateModelToDownButton = wmgr.createWindow("but/topButton", "CEGUIDemo/translateModelToDownButton");

translateModelToDownButton->setRotation( CEGUI::Vector3(0,0,180) );

translateModelToDownButton->setSize(CEGUI::UVector2(CEGUI::UDim(0.07, 0), CEGUI::UDim(0.07, 0)));

translateModelToDownButton->setPosition(CEGUI::UVector2(CEGUI::UDim(0.7, 0), CEGUI::UDim(0.62, 0)));

//Кнопка перемещения модели вверх

rotateRightModelButton = wmgr.createWindow("but/rotateButton", "CEGUIDemo/rotateRightModelButton");

rotateRightModelButton->setSize(CEGUI::UVector2(CEGUI::UDim(0.07, 0), CEGUI::UDim(0.07, 0)));

rotateRightModelButton->setPosition(CEGUI::UVector2(CEGUI::UDim(0.515, 0), CEGUI::UDim(0.6, 0)));

//Кнопка перемещения модели вверх

rotateLeftModelButton = wmgr.createWindow("but/rotateButton", "CEGUIDemo/rotateLeftModelButton");

rotateLeftModelButton->setRotation( CEGUI::Vector3(0,180,0) );

rotateLeftModelButton->setSize(CEGUI::UVector2(CEGUI::UDim(0.07, 0), CEGUI::UDim(0.07, 0)));

rotateLeftModelButton->setPosition(CEGUI::UVector2(CEGUI::UDim(0.5, 0), CEGUI::UDim(0.7, 0)));

//Кнопка возвращения на главное окно

backMainWindowButton = wmgr.createWindow("OgreTray/Button", "CEGUIDemo/backMainWindowButton");

wchar\_t saveText[100] = L"Сохранить";

backMainWindowButton->setText(encodeRussian(saveText) );

backMainWindowButton->setSize(CEGUI::UVector2(CEGUI::UDim(0.15, 0), CEGUI::UDim(0.05, 0)));

backMainWindowButton->setPosition(CEGUI::UVector2(CEGUI::UDim(0.85, 0), CEGUI::UDim(0, 0)));

//настройки комнаты

//Кнопка увеличения скейла

scaleRoomPlusButton = wmgr.createWindow("but/scalePlusButton", "CEGUIDemo/scaleRoomPlusButton");

scaleRoomPlusButton->setSize(CEGUI::UVector2(CEGUI::UDim(0.07, 0), CEGUI::UDim(0.07, 0)));

scaleRoomPlusButton->setPosition(CEGUI::UVector2(CEGUI::UDim(0.15, 0), CEGUI::UDim(0.6, 0)));

//Кнопка уменьшения скейла

scaleRoomMinusButton = wmgr.createWindow("but/scaleMinusButton", "CEGUIDemo/scaleRoomMinusButton");

scaleRoomMinusButton->setSize(CEGUI::UVector2(CEGUI::UDim(0.07, 0), CEGUI::UDim(0.07, 0)));

scaleRoomMinusButton->setPosition(CEGUI::UVector2(CEGUI::UDim(0.25, 0), CEGUI::UDim(0.6, 0)));

//типса настроек

tipsSettingText = wmgr.createWindow("OgreTray/StaticText", "CEGUIDemo/tipsTextSettings");

tipsSettingText->setVisible(true);

tipsSettingText->setSize(CEGUI::UVector2(CEGUI::UDim(1, 0), CEGUI::UDim(0.07, 0)));

tipsSettingText->setPosition(CEGUI::UVector2(CEGUI::UDim(0, 0), CEGUI::UDim(0.93, 0)));

wchar\_t tipSettingText[150] = L"Режим настроек: удерживайте ПКМ для поворота камеры. WASD - двигать камеру. ЛКМ - выбрать.";

tipsSettingText->setText( encodeRussian(tipSettingText) );

//--------текст типсы главного экрана настроек

tipsText = wmgr.createWindow("OgreTray/StaticText", "CEGUIDemo/tipsText");

tipsText->setVisible(true);

tipsText->setSize(CEGUI::UVector2(CEGUI::UDim(1, 0), CEGUI::UDim(0.07, 0)));

tipsText->setPosition(CEGUI::UVector2(CEGUI::UDim(0, 0), CEGUI::UDim(0.93, 0)));

wchar\_t tipText[150] = L"Режим поворота камеры: удерживайте ПКМ для поворота камеры. WASD - двигать камеру. ЛКМ - выбрать.";

tipsText->setText( encodeRussian(tipText) );

//--------текст типсы масштаба зала

tipsRoomText = wmgr.createWindow("OgreTray/StaticText", "CEGUIDemo/tipsRoomText");

tipsRoomText->setVisible(true);

tipsRoomText->setSize(CEGUI::UVector2(CEGUI::UDim(0.22, 0), CEGUI::UDim(0.07, 0)));

tipsRoomText->setPosition(CEGUI::UVector2(CEGUI::UDim(0.12, 0), CEGUI::UDim(0.5, 0)));

wchar\_t tipRoomText[150] = L"Масштабировать зал";

tipsRoomText->setText( encodeRussian(tipRoomText) );

//--------текст типсы манипуляции объекта

tipsObjectText = wmgr.createWindow("OgreTray/StaticText", "CEGUIDemo/tipsObjectText");

tipsObjectText->setVisible(true);

tipsObjectText->setSize(CEGUI::UVector2(CEGUI::UDim(0.3, 0), CEGUI::UDim(0.07, 0)));

tipsObjectText->setPosition(CEGUI::UVector2(CEGUI::UDim(0.535, 0), CEGUI::UDim(0.5, 0)));

wchar\_t tipObjectText[150] = L"Манипулирование объектом";

tipsObjectText->setText( encodeRussian(tipObjectText) );

//--------крепим кнопки к виджетам

sheetMgr->addChildWindow(sheetMainWindow);

sheetMgr->addChildWindow(sheetMainMenu);

sheetMainWindow->addChildWindow(buttonQuit);

sheetMainWindow->addChildWindow(buttonCameraSetPosition);

sheetMainWindow->addChildWindow(textbox1);

sheetMainWindow->addChildWindow(menuButton);

sheetMainWindow->addChildWindow(tipsText);

sheetMainMenu->addChildWindow(scaleRoomPlusButton);

sheetMainMenu->addChildWindow(scaleRoomMinusButton);

sheetMainMenu->addChildWindow(scalePlusButton);

sheetMainMenu->addChildWindow(scaleMinusButton);

sheetMainMenu->addChildWindow(translateModelToLeftButton);

sheetMainMenu->addChildWindow(translateModelToRightButton);

sheetMainMenu->addChildWindow(translateModelToTopButton);

sheetMainMenu->addChildWindow(translateModelToBackButton);

sheetMainMenu->addChildWindow(translateModelToUpButton);

sheetMainMenu->addChildWindow(translateModelToDownButton);

sheetMainMenu->addChildWindow(rotateRightModelButton);

sheetMainMenu->addChildWindow(rotateLeftModelButton);

sheetMainMenu->addChildWindow(backMainWindowButton);

sheetMainMenu->addChildWindow(checkShadow);

sheetMainMenu->addChildWindow(textShadow);

sheetMainMenu->addChildWindow(tipsSettingText);

sheetMainMenu->addChildWindow(tipsRoomText);

sheetMainMenu->addChildWindow(tipsObjectText);

sheetMainMenu->hide();

//--------обрабатываем нажатия кнопок

buttonQuit->subscribeEvent(CEGUI::PushButton::EventClicked, CEGUI::Event::Subscriber(&BasicApp::quit, this));

buttonCameraSetPosition->subscribeEvent(CEGUI::PushButton::EventClicked, CEGUI::Event::Subscriber(&BasicApp::setDefaultPosition, this));

menuButton->subscribeEvent(CEGUI::PushButton::EventClicked, CEGUI::Event::Subscriber(&BasicApp::eventMenuButton, this));

backMainWindowButton->subscribeEvent(CEGUI::PushButton::EventClicked, CEGUI::Event::Subscriber(&BasicApp::eventBackWindow, this));

Ogre::Vector3 scaleRoomVector(0, 0, 0);

for(int i = 0; i < myVector.size(); i++)

{

Ogre::Entity\* lEntity = mSceneMgr->createEntity(myVector[i], myVector[i] + ".mesh" );

lNode = mSceneMgr->getRootSceneNode()->createChildSceneNode(myVector[i]);

lNode->attachObject(lEntity);

const char \* nameModel = nullptr; //преобразования в const char для xml

char stringToChar [100];

std::strcpy( stringToChar, myVector[i].c\_str() );

nameModel = stringToChar;

Ogre::Vector3 locationVector(0, 0, 0);//получаем координаты модельки

Ogre::Vector3 scaleVector(0, 0, 0);//её скейл

double rotate = 0.0;//и поворот

configureXmlSettingsD(nameModel, locationVector, scaleVector, rotate, scaleRoomVector);

lNode->setPosition(locationVector);

lNode->setScale(scaleVector);

lNode->yaw(Ogre::Degree(rotate));

removeFromXmlElderData();

}

room = mSceneMgr->createEntity("Cube.001.mesh");

roomNode = mSceneMgr->getRootSceneNode()->createChildSceneNode("room");

roomNode->attachObject(room);

roomNode->setPosition(500,450,200);

roomNode->setScale(scaleRoomVector);

//roomNode->setScale(50,25,50);

static\_cast<CEGUI::Checkbox\*>(checkShadow)->setSelected(true);

mSceneMgr->setShadowTechnique(Ogre::SHADOWTYPE\_STENCIL\_ADDITIVE);

tipsEnabled = true;

CEGUI::System::getSingleton().setGUISheet(sheetMgr);

}

void BasicApp::checkShadows()

{

if ( static\_cast<CEGUI::Checkbox\*>(checkShadow)->isSelected() )

{

mSceneMgr->setShadowTechnique(Ogre::SHADOWTYPE\_STENCIL\_ADDITIVE);

}

else if ( !static\_cast<CEGUI::Checkbox\*>(checkShadow)->isSelected() )

{

mSceneMgr->setShadowTechnique(Ogre::SHADOWTYPE\_NONE);

}

}

/\*

bool BasicApp::releaseCam()

{

return static\_cast<CEGUI::Checkbox\*>(releaseCamera)->isSelected();

}

\*/

CEGUI::String BasicApp::encodeRussian(wchar\_t string[100])

{

wchar\_t strw[100] = L"";

wcscat(strw, string);

CEGUI::String str;

for (int i=0; i < wcslen(strw); i++)

{

str += strw[i];

}

return str;

}

bool BasicApp::eventBackWindow(const CEGUI::EventArgs &e)

{

configureXmlSettings();

sheetMainMenu->hide();

sheetMainMenu->disable();

sheetMainMenu->deactivate();

sheetMainWindow->show();

sheetMainWindow->activate();

sheetMainWindow->enable();

for(int i = 0; i < myVector.size(); i++)

{

mSceneMgr->getSceneNode(myVector[i])->showBoundingBox(false); //убираем выделение со всех моделек на сцене

}

textbox1->setVisible(false);

return true;

}

void BasicApp::configureXmlSettingsD(const char \* nameModel, Ogre::Vector3 &locationVector, Ogre::Vector3 &scaleVector, double &rotate, Ogre::Vector3 &scaleRoomVector)

{

file\_settings->LoadFile();

bool rewrite = false;

bool addNew = false;

TiXmlElement\* xml\_name\_model = new TiXmlElement(nameModel);

TiXmlNode\* findModel = file\_settings->FirstChildElement(nameModel); //проверяем, есть ли моделька из вектора в xml

if (findModel == nullptr)

{

addNew = true;//если нет, добавляем новую

}

if (!addNew)//иначе обновляем атрибуты (потом можно переписать красивее)

{

for( xml\_name\_model = file\_settings->FirstChildElement(nameModel); xml\_name\_model != NULL; xml\_name\_model = xml\_name\_model->NextSiblingElement(nameModel) )

{

if ( xml\_name\_model != nullptr)

{

rewrite = true;

break;

}

}

}

if (rewrite)

{

double xPos = 0.0;

double yPos = 0.0;

double zPos = 0.0;

double xScale = 0.0;

double yScale = 0.0;

double zScale = 0.0;

double rotateModel = 0.0;

xml\_name\_model->QueryDoubleAttribute("xPosition", &xPos);

xml\_name\_model->QueryDoubleAttribute("yPosition", &yPos);

xml\_name\_model->QueryDoubleAttribute("zPosition", &zPos);

locationVector.x = xPos;

locationVector.y = yPos;

locationVector.z = zPos;

xml\_name\_model->QueryDoubleAttribute("xScale", &xScale);

xml\_name\_model->QueryDoubleAttribute("yScale", &yScale);

xml\_name\_model->QueryDoubleAttribute("zScale", &zScale);

scaleVector.x = xScale;

scaleVector.y = yScale;

scaleVector.z = zScale;

xml\_name\_model->QueryDoubleAttribute("rotate", &rotateModel);

rotate = rotateModel;

}

if (addNew)

{

file\_settings->LinkEndChild(xml\_name\_model);

std::srand( std::time(NULL) );

locationVector.x = 0;

locationVector.y = 0;

locationVector.z = std::rand()%300;

xml\_name\_model->SetDoubleAttribute("xPosition", locationVector.x );

xml\_name\_model->SetDoubleAttribute("yPosition", locationVector.y );

xml\_name\_model->SetDoubleAttribute("zPosition", locationVector.z );

scaleVector.x = 1;

scaleVector.y = 1;

scaleVector.z = 1;

xml\_name\_model->SetDoubleAttribute("xScale", scaleVector.x);

xml\_name\_model->SetDoubleAttribute("yScale", scaleVector.y);

xml\_name\_model->SetDoubleAttribute("zScale", scaleVector.z);

rotate = 0.0;

xml\_name\_model->SetDoubleAttribute("rotate", rotate);

file\_settings->SaveFile();

}

file\_room\_settings->LoadFile();

TiXmlElement\* xml\_name\_room = new TiXmlElement("room");

xml\_name\_room = file\_room\_settings->FirstChildElement("room");

double xScale = 0.0;

double yScale = 0.0;

double zScale = 0.0;

xml\_name\_room->QueryDoubleAttribute("xScale", &xScale);

xml\_name\_room->QueryDoubleAttribute("yScale", &yScale);

xml\_name\_room->QueryDoubleAttribute("zScale", &zScale);

scaleRoomVector.x = xScale;

scaleRoomVector.y = yScale;

scaleRoomVector.z = zScale;

}

void BasicApp::configureXmlSettings()

{

file\_settings->LoadFile();

for(int i = 0; i < myVector.size(); i++)

{

const char \* nameModel = nullptr;//преобразования в const char для xml

char stringToChar [100];

std::strcpy( stringToChar, myVector[i].c\_str() );

nameModel = stringToChar;

TiXmlElement\* xml\_name\_model = nullptr;

xml\_name\_model = file\_settings->FirstChildElement(nameModel);

xml\_name\_model->SetDoubleAttribute("xPosition", mSceneMgr->getSceneNode(nameModel)->getPosition().x);

xml\_name\_model->SetDoubleAttribute("yPosition", mSceneMgr->getSceneNode(nameModel)->getPosition().y);

xml\_name\_model->SetDoubleAttribute("zPosition", mSceneMgr->getSceneNode(nameModel)->getPosition().z);

xml\_name\_model->SetDoubleAttribute("xScale", mSceneMgr->getSceneNode(nameModel)->getScale().x);

xml\_name\_model->SetDoubleAttribute("yScale", mSceneMgr->getSceneNode(nameModel)->getScale().y);

xml\_name\_model->SetDoubleAttribute("zScale", mSceneMgr->getSceneNode(nameModel)->getScale().z);

xml\_name\_model->SetDoubleAttribute("rotate", mSceneMgr->getSceneNode(nameModel)->getOrientation().getYaw().valueDegrees() );

file\_settings->SaveFile();

}

file\_room\_settings->LoadFile();

TiXmlElement\* xml\_name\_model = nullptr;

xml\_name\_model = file\_room\_settings->FirstChildElement("room");

xml\_name\_model->SetDoubleAttribute("xScale", mSceneMgr->getSceneNode("room")->getScale().x);

xml\_name\_model->SetDoubleAttribute("yScale", mSceneMgr->getSceneNode("room")->getScale().y);

xml\_name\_model->SetDoubleAttribute("zScale", mSceneMgr->getSceneNode("room")->getScale().z);

file\_room\_settings->SaveFile();

}

void BasicApp::removeFromXmlElderData()

{

file\_settings->LoadFile();

std::vector<std::string> notExistsModels; //вектор для дальнейшего удаления неиспользованных моделек из xml

//считываем все имена моделек с xml в вектор

TiXmlNode\* xml\_name = nullptr;

for( xml\_name = file\_settings->FirstChildElement(); xml\_name != NULL; xml\_name = xml\_name->NextSiblingElement() )

{

std::string name = xml\_name->Value();

notExistsModels.push\_back(name);

}

if ( notExistsModels.size() >= myVector.size() ) //костыль (падало при запуске при добавлении сразу 3 новых моделек в конфиг

//поэтому ждём пока подгрузит всё из конфига)

{

//удаляем лишние элементы из вектора, которые не используются в приложении

for(int i = 0; i < myVector.size(); i++ )

{

//ищем элемент подгрузочного вектора в векторе из xml

std::vector<std::string>::iterator it = std::find(notExistsModels.begin(), notExistsModels.end(), myVector[i]);

notExistsModels.erase( it );

std::vector<std::string>(notExistsModels).swap(notExistsModels); //для освобождения памяти

}

//теперь удаляем из xml всё, что осталось в векторе notExists

for(int i = 0; i < notExistsModels.size(); i++ )

{

const char \* nameModel = nullptr; //преобразования в const char для xml

char stringToChar [100];

std::strcpy( stringToChar, notExistsModels[i].c\_str() );

nameModel = stringToChar;

TiXmlNode\* xml\_delete = file\_settings->FirstChildElement(nameModel);

file\_settings->RemoveChild(xml\_delete);

file\_settings->SaveFile();

}

}

}

bool BasicApp::eventMenuButton(const CEGUI::EventArgs &e)

{

sheetMainWindow->hide();

sheetMainWindow->disable();

sheetMainWindow->deactivate();

sheetMainMenu->show();

sheetMainMenu->activate();

sheetMainMenu->enable();

return true;

}

bool BasicApp::quit(const CEGUI::EventArgs &e)

{

mShutdown = true;

return true;

}

bool BasicApp::setDefaultPosition(const CEGUI::EventArgs &e)

{

goDefaultPosition = !goDefaultPosition;

textbox1->setVisible(false);

for(int i = 0; i < myVector.size(); i++)

{

mSceneMgr->getSceneNode(myVector[i])->showBoundingBox(false); //убираем выделение со всех моделек на сцене

}

timer1.reset();

wchar\_t cameraText[100] = L"Остановить камеру";

if (buttonCameraSetPosition->getText() == encodeRussian(cameraText) )

{

tipsEnabled = !tipsEnabled;

}

return true;

}

void BasicApp::destroyScene()

{

}

void BasicApp::createFrameListener()

{

Ogre::LogManager::getSingletonPtr()->logMessage("\*\*\* Initializing OIS \*\*\*");

OIS::ParamList pl;

size\_t windowHnd = 0;

std::ostringstream windowHndStr;

mWindow->getCustomAttribute("WINDOW", &windowHnd);

windowHndStr << windowHnd;

pl.insert(std::make\_pair(std::string("WINDOW"), windowHndStr.str()));

mInputMgr = OIS::InputManager::createInputSystem(pl);

mKeyboard = static\_cast<OIS::Keyboard\*>(

mInputMgr->createInputObject(OIS::OISKeyboard, true));

mMouse = static\_cast<OIS::Mouse\*>(

mInputMgr->createInputObject(OIS::OISMouse, true));

mKeyboard->setEventCallback(this);

mMouse->setEventCallback(this);

windowResized(mWindow);

Ogre::WindowEventUtilities::addWindowEventListener(mWindow, this);

mRoot->addFrameListener(this);

Ogre::LogManager::getSingletonPtr()->logMessage("Finished");

}

void BasicApp::createViewports()

{

Ogre::Viewport\* vp = mWindow->addViewport(mCamera);

vp->setBackgroundColour(Ogre::ColourValue(0, 0, 0));

mCamera->setAspectRatio(

Ogre::Real(vp->getActualWidth()) /

Ogre::Real(vp->getActualHeight()));

}

void BasicApp::setupResources()

{

Ogre::ConfigFile cf;

cf.load(mResourcesCfg);

Ogre::String secName, typeName, archName;

Ogre::ConfigFile::SectionIterator secIt = cf.getSectionIterator();

while (secIt.hasMoreElements())

{

secName = secIt.peekNextKey();

Ogre::ConfigFile::SettingsMultiMap\* settings = secIt.getNext();

Ogre::ConfigFile::SettingsMultiMap::iterator setIt;

for (setIt = settings->begin(); setIt != settings->end(); ++setIt)

{

typeName = setIt->first;

archName = setIt->second;

Ogre::ResourceGroupManager::getSingleton().addResourceLocation(

archName, typeName, secName);

//преобразуем строку огре в char

std::string ogreToString = static\_cast<std::string>(archName);

char stringToChar [100];

std::strcpy( stringToChar, ogreToString.c\_str() );

if ( std::strstr(stringToChar, ".zip") )

{

std::memset(result, 0, sizeof(result)); //очистим result

int len = strlen(stringToChar); //длина считанной строки из конфига

int i = 0;

//добираемся до первого слеша с конца

for(i = len; i > 0 ;i--)

{

if ( stringToChar[i-1] == '/')

{

break;

}

}

int res = 0;

//вырезаем имя меша

for(int j = i; j < len ;j++)

{

result[res++] = stringToChar[j]; //записываем название меша без .mesh

if ( stringToChar[j+1] == '.')

{

break;

}

}

myVector.push\_back( std::string(result) ); //заносим название в вектор

}

}

}

}

void BasicApp::createResourceListener()

{

}

void BasicApp::loadResources()

{

Ogre::ResourceGroupManager::getSingleton().initialiseAllResourceGroups();

}

bool BasicApp::setupCEGUI()

{

Ogre::LogManager::getSingletonPtr()->logMessage("\*\*\* Initializing CEGUI \*\*\*");

mRenderer = &CEGUI::OgreRenderer::bootstrapSystem();

CEGUI::Imageset::setDefaultResourceGroup("Imagesets");

CEGUI::Font::setDefaultResourceGroup("Fonts");

CEGUI::Scheme::setDefaultResourceGroup("Schemes");

CEGUI::WidgetLookManager::setDefaultResourceGroup("LookNFeel");

CEGUI::WindowManager::setDefaultResourceGroup("Layouts");

CEGUI::SchemeManager::getSingleton().create("OgreTray.scheme");

CEGUI::SchemeManager::getSingleton().create("WindowsLook.scheme");

CEGUI::SchemeManager::getSingleton().create("TaharezLook.scheme");

CEGUI::SchemeManager::getSingleton().create("VanillaSkin.scheme");

CEGUI::SchemeManager::getSingleton().create("Novy\_Kholst.scheme");

CEGUI::SchemeManager::getSingleton().create("but.scheme");

CEGUI::FontManager::getSingleton().create("DejaVuSans-10.font");

CEGUI::System::getSingleton().setDefaultFont("DejaVuSans-10");

CEGUI::System::getSingleton().setDefaultMouseCursor("WindowsLook", "MouseArrow");

CEGUI::MouseCursor::getSingleton().setImage( CEGUI::System::getSingleton().getDefaultMouseCursor());

Ogre::LogManager::getSingletonPtr()->logMessage("Finished");

return true;

}

void BasicApp::handleCameraCollision() //метод ограничения камеры (не летаем за текстуры)

{

Ogre::Vector3 camPos = mCamera->getPosition();

Ogre::Ray camRay(

Ogre::Vector3(camPos.x, 5000.0, camPos.z),

Ogre::Vector3::NEGATIVE\_UNIT\_Y);

Ogre::TerrainGroup::RayResult result = mTerrainGroup->rayIntersects(camRay);

if (result.terrain)

{

Ogre::Real terrainHeight = result.position.y;

if (camPos.y < (terrainHeight + 10.0))

mCamera->setPosition(camPos.x, terrainHeight + 10.0, camPos.z);

}

}

#if OGRE\_PLATFORM == OGRE\_PLATFORM\_WIN32

#define WIN32\_LEAN\_AND\_MEAN

#include "windows.h"

#endif

#ifdef \_\_cplusplus

extern "C" {

#endif

#if OGRE\_PLATFORM == OGRE\_PLATFORM\_WIN32

INT WINAPI WinMain(HINSTANCE hInst, HINSTANCE, LPSTR strCmdLine, INT)

#else

int main(int argc, char \*argv[])

#endif

{

BasicApp app;

try

{

app.go();

}

catch(Ogre::Exception& e)

{

#if OGRE\_PLATFORM == OGRE\_PLATFORM\_WIN32

MessageBox(

NULL,

e.getFullDescription().c\_str(),

"An exception has occured!",

MB\_OK | MB\_ICONERROR | MB\_TASKMODAL);

#else

std::cerr << "An exception has occured: " <<

e.getFullDescription().c\_str() << std::endl;

#endif

}

return 0;

}

#ifdef \_\_cplusplus

}

#endif

textMgr.h

#include <cstring>

#include <vector>

#include <fstream>

#include <CEGUI.h>

class textMgr

{

public:

textMgr();

~textMgr();

std::vector<std::string> idTextVector; //идентификатор текста модельки, сравниваем по имени

std::vector<std::string> dataTextVector; //описание модели

std::string getText(std::string id);

void setText(std::string id, std::string data);

wchar\_t\* getFromFile(std::string id);

private:

};

textMgr.cpp

#include "textMgr.h"

textMgr::textMgr()

{

}

textMgr::~textMgr()

{

}

std::string textMgr::getText(std::string id)

{

for(int i = 0; i < idTextVector.size(); i++)

{

if ( idTextVector[i].length() == id.length() )

{

for(int j = 0; j < id.length(); j++)

{

if (id[j] != idTextVector[i][j])

{

return "Add text for model!";

}

else

{

return dataTextVector[0];

}

}

}

else

{

return "Add text for model!";

}

}

}

wchar\_t\* textMgr::getFromFile(std::string id)

{

setlocale(LC\_ALL, "Russian");

id += ".txt"; //добавляем .txt к переданному мешу

//CEGUI::String data; //возвращаемая строка

wchar\_t data[2000] = L"";

std::string path = "media\\text\\";

path += id; //к пути прибаляем имя меша + .txt

std::FILE\* file;

file = std::fopen(path.c\_str(), "r");

char line[100];

if (file == NULL) //если файла описания нет, просим добавить

{

wchar\_t info[50] = L"Текст не найден!";

wcscat(data, info);

}

else

{

while ( !std::feof(file) )

{

if ( std::fgets(line,100,file) )

{

wchar\_t temp[250] = L"";

mbstowcs(temp,line,250); //конвертируем из char в wchar\_t

//temp[250] = (wchar\_t)line;

wcscat(data,temp);

}

}

fclose(file);

}

CEGUI::String info;

for (int i=0; i < wcslen(data); i++)

{

info += data[i];

}

return data;

}

Размещено на Allbest.ru