Технология сборки проекта:

Инициализация проекта:

Есть две внешние библиотеки:

1. Из проекта World3D
2. OgreExtension

Их необходимо включить в папку проекта.

В проекте необходимо создать файл resource.cfg, в котором будут прописаны пути к ресурсам проекта.

Пример файла resource.cfg:

FileSystem=./rsrc/

Zip = ./OgreExtension/Resources/RadioButton.zip

Zip = ./OgreExtension/Resources/HorisontalSlider.zip

Zip = ./OgreExtension/Resources/VerticalSlider.zip

Zip = ./OgreExtension/Resources/CheckBox.zip

Zip = ./OgreExtension/Resources/Button.zip

Zip = ./OgreExtension/Resources/StandardPrimitives.zip

Zip = .//OgreExtension/Resources/Fonts.zip

Zip = ./OgreExtension/Resources/Colors.zip

С помощью данных строк Ogre3d будет искать ресурсы в указанных папках, при это он ищет не рекурсивно.

При запуске Ogre в 1 раз, создаётся Ogre.cfg.

В итоге папка проекта:

Img/

OgreExtension/

Rsrc/

resource.cfg

Ogre.cfg

Есть следующие типы ресурсов:

.mesh

.material

.particle

.overlay

.scene

Рассмотрим .material:

material Land\_OgreMax

{

receive\_shadows on

technique Map#6

{

pass Map#7

{

ambient 0.0784314 0.909804 0.0352941 1

diffuse 0.0784314 0.909804 0.0352941 1

specular 0 0 0 1 10

texture\_unit Map#8

{

texture\_alias Map#8

texture grass2.jpg

filtering linear linear linear

colour\_op modulate

scale 0.2 0.2

scroll 0 0

rotate 0

tex\_address\_mode wrap

}

}

}

}

Таким образом можно описать материал и включить его в проекте, указав название материала Land\_OgreMax в имидже.

Также материал полностью можно создать через имидж.

Подробнее о материалах в разделе Материалы.

Другое дело частицы:

particle\_system WaterDrop

{

material WaterDrop

particle\_width 0.2

particle\_height 0.2

cull\_each true

quota 5000

billboard\_type point

emitter Ellipsoid

{

height 1000

width 1000

inner\_height 0.9

inner\_width 0.9

emission\_rate 500

direction 1, -1, 0

velocity 50

}

}

Имиджа для описания частиц нет, так что необходимо создавать файл с расширением. particle, а в имидже ParticleSystem указывать название системы частиц.

Импорт 3d объектов:

Ogre3d читает только свой формат .mesh, .sceleton

Можно конвертировать любой формат с помощью плагина 3d редактора (3dsMax, Blender). Или использовать Assimp: <https://github.com/assimp/assimp>.

Рассмотрим 1вариант:

От 3ds MAX 9 до 3ds MAX 2018:

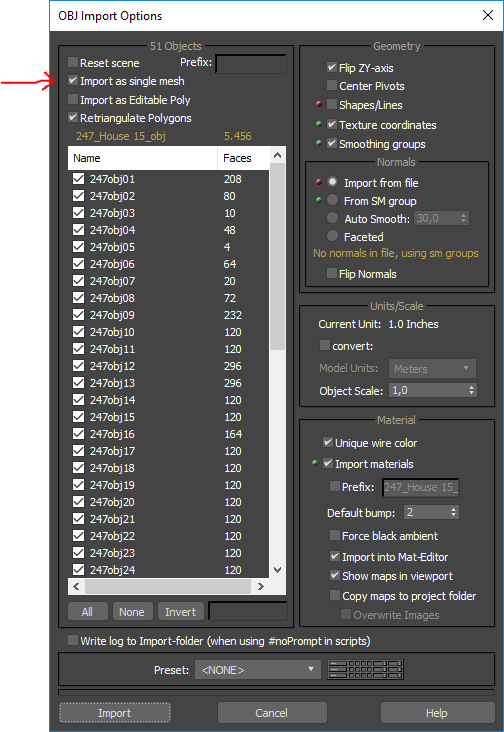
Зайти на сайт для ознакомления с информацией:

<http://wiki.ogre3d.org/Easy+Ogre+Exporter>

Перейти по ссылке [Install for Max 2011 to 2018](http://www.openspace3d.com/downloads/EasyOgreExporter) и скачать файл установки.

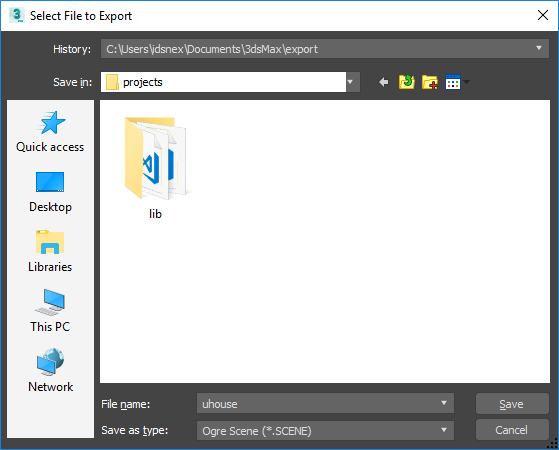
Во время установки версия 3ds Max должна определится сама.

Открываем 3ds Max редактор и импортируем модель (во время импорта есть возможно импортировать как 1 меш)

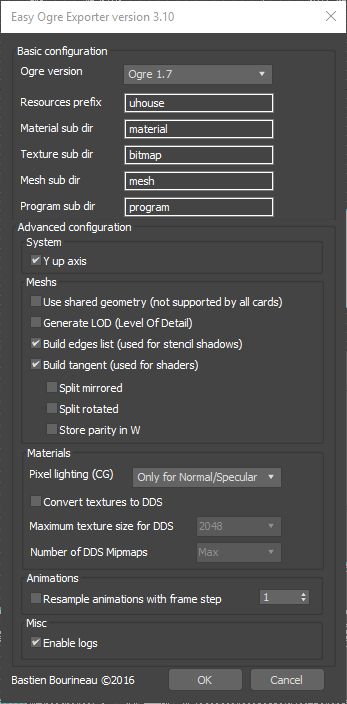


Далее заходим в File -> Export -> Export…

Выбираем тип Ogre Scene

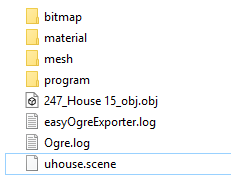


Затем выбираем версию Ogre (В stratum скорее всего Ogre 1.7) и настраиваем Export:



И нажимаем ок. Модель экспортирована в scene, откуда можно получить mesh.

Директория выглядит таким образом:



Для загрузки объекта нам нужен mesh и материал. Они расположены в соответствующих папках. Кладём данные файлы в rsrc проекта (папка ресурсов).

Получить модели можно в данных ресурсах:

<https://archive3d.net/>

<https://free3d.com/3d-models/>

Создание сцены:

Для работы Ogre3d необходимы имиджи из стандартной библиотеки:

Ogre\_Root

Ogre\_Scene



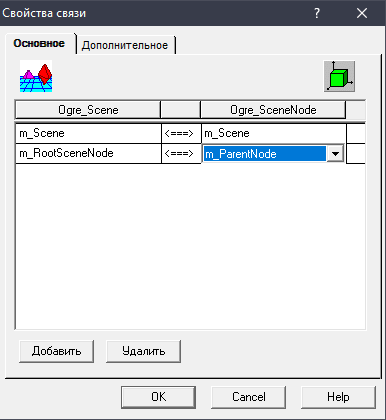
В Ogre\_Root прописываем путь к файлу ресурсов:

m\_ResourceConfigFileName = resource.cfg

В Ogre\_Scene можно указать материал для SkyBox – неба.

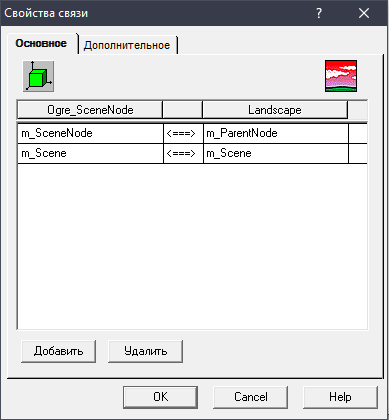
Ландшафт

Создадим глобальный узел для указана глобальных позиций, размеров и поворота с помощью SceneNode. Соединим со сценой таким оразом:

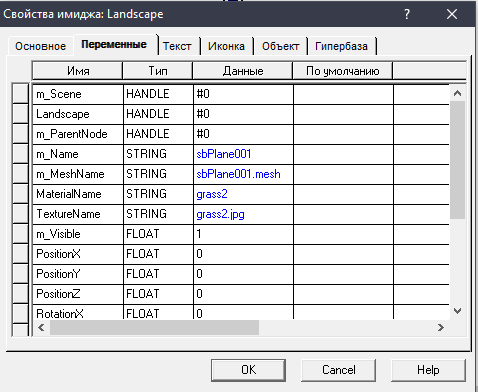


Каждый дочерний узел соединяется с предком с помощью поля ParentNode.

Теперь вставим имидж из проекта – LandScape, и соединим с ScenrNode:



В свойствах имиджа укажем имя и название ресурса mesh и материал:

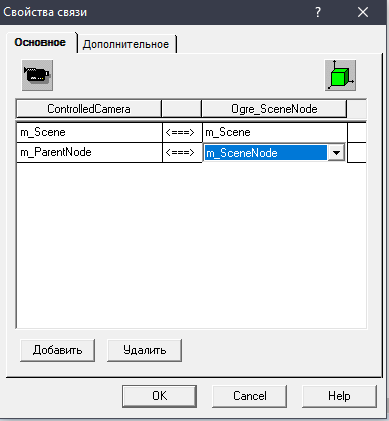


Материал можно загрузить через файлы ресурсов, указав имя материала (более правильный способ).

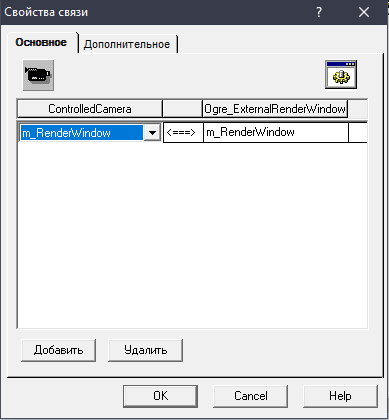
Создание управляемой камеры:

В проекте есть имидж ControlledCamera с помощью ней можно вывести на экран изображение.

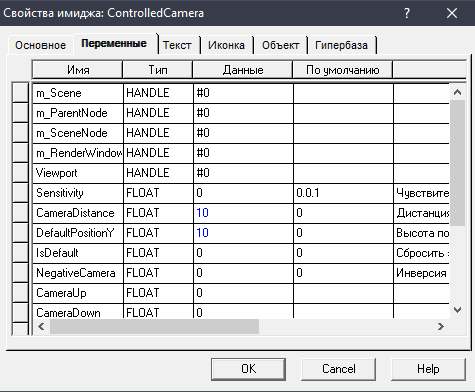
Соединим её с глобиным узлом сцены:



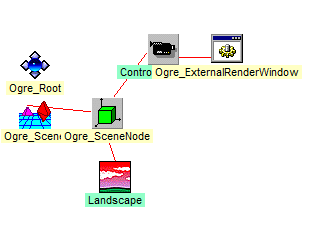
Из стандартной библиотеки вставим внешнее окно рендеринга (в нём своя обработка ввода) Ogre\_ExternalRenderWindow: и соединим с камерой:



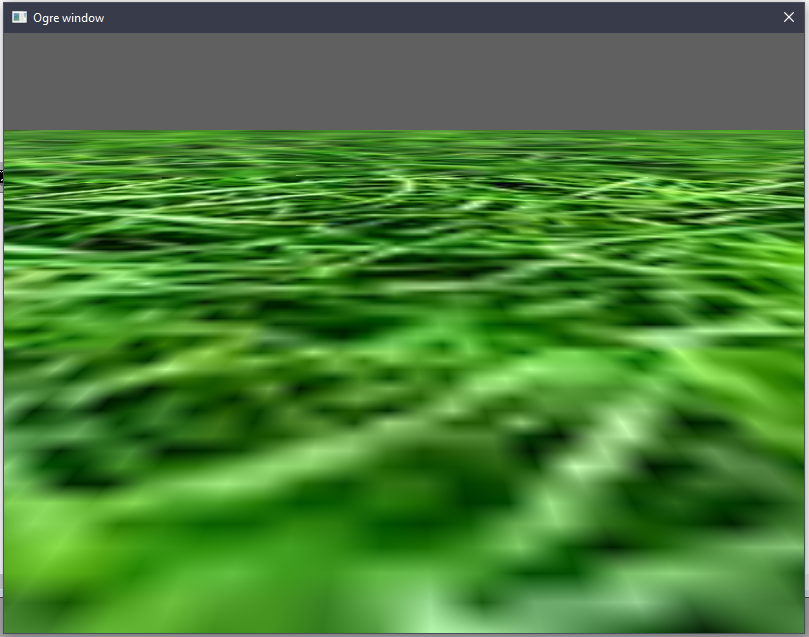
Временно поставим настройки камеры такие:



В итоге схема:



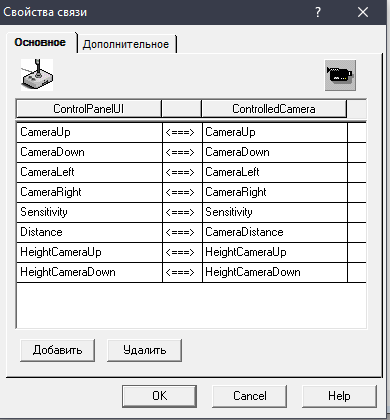
Запускаем проект и видим ландшафт:



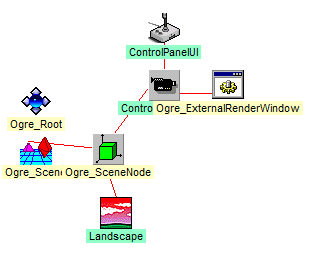
**Управление**

Вставим из проекта ControlPanelUI:

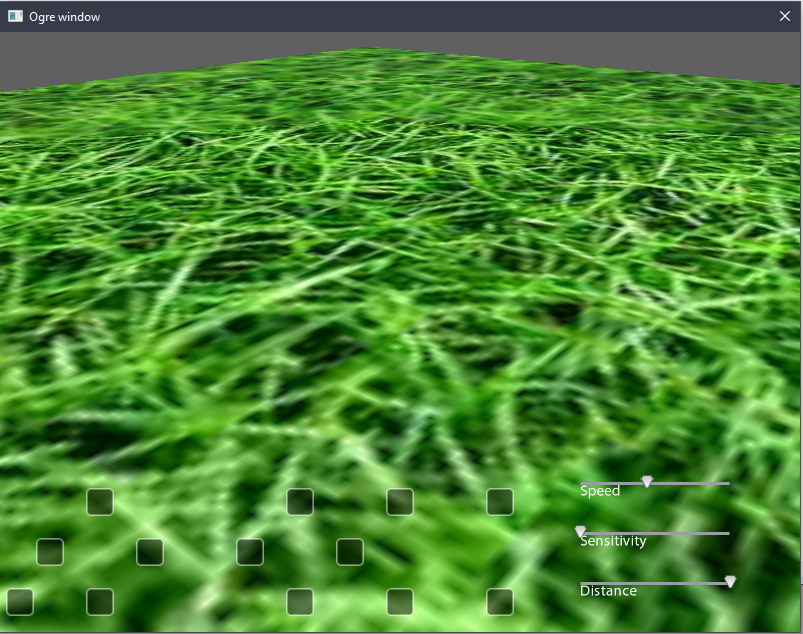
И свяжем с камерой:



В итоге связи:



Запустим проект и увидим панель управление (работает всё кроме перемещения):



Персонаж

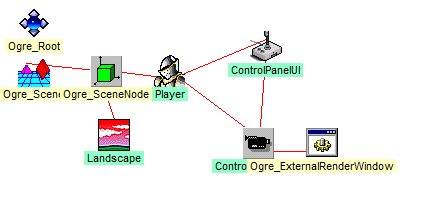
На данный момент у нас есть 2 персонажа с анимацией:

Boy

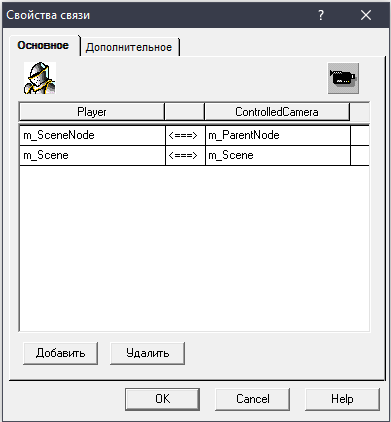
Sinbad

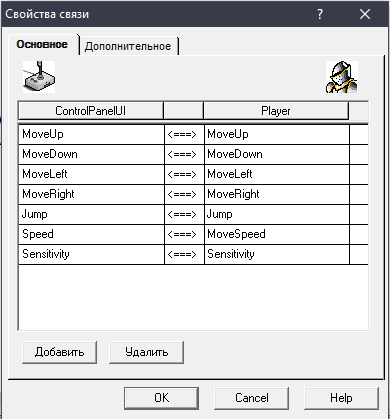
Вставим второго, не забыв указать путь к архиву.

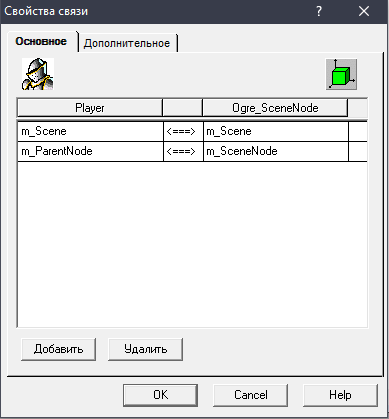
Вставим имидж Player из проекта и соединим имиджи таким образом:



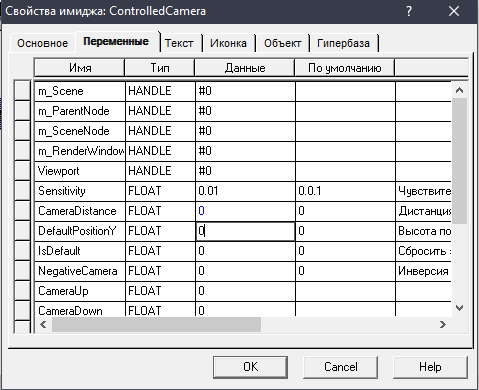
Установим следующие связи:

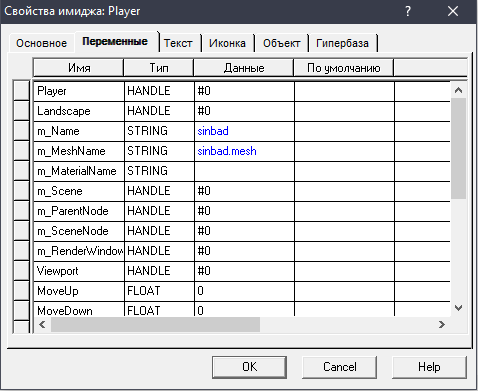


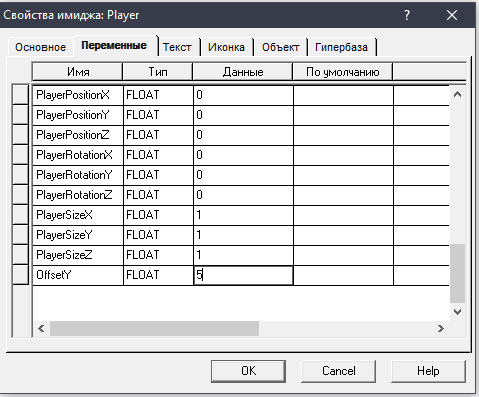




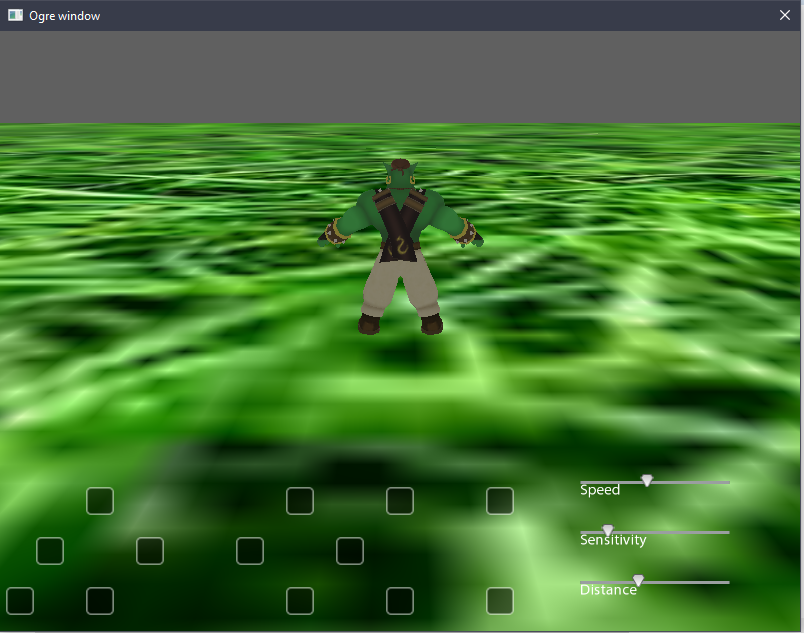
Установим свойств имиджей:







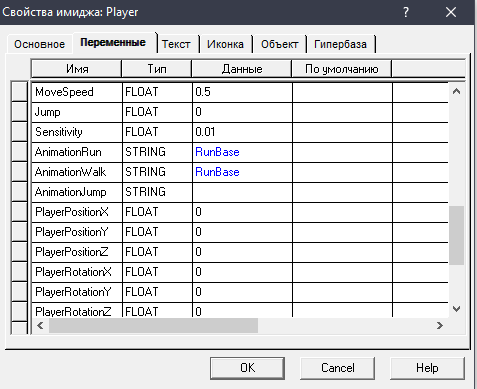
В итоге получим картину:



Персонаж теперь управляем и натравливается сквозь ландшафт.

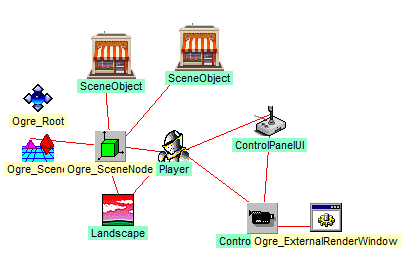
Добавим анимации, которые можно получить с помощью ресурса .sceleton, который в свою очередь получает из анимации импортированной модели.

Указываем в свойствах имиджа названия:



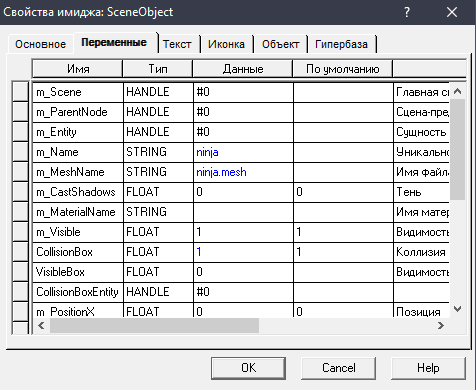
Добавление других объектов:

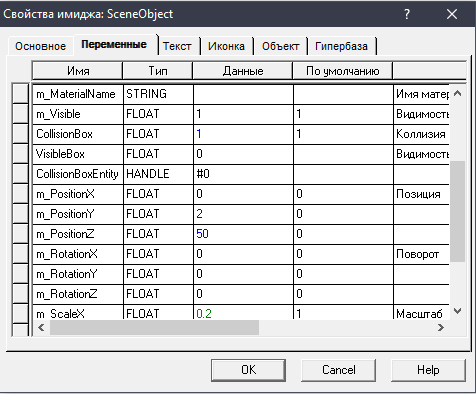
Создадим такую схему:

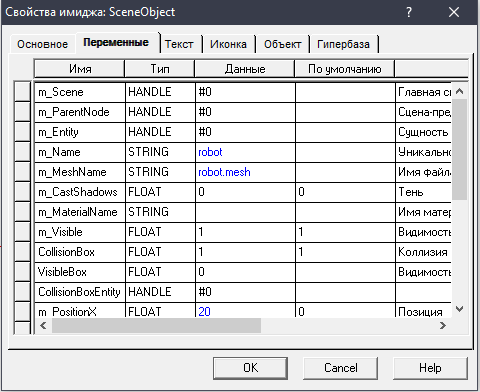


Соединим имиджи к глобальному узлу также, как ландшафт:

Изменим следующие свойства:







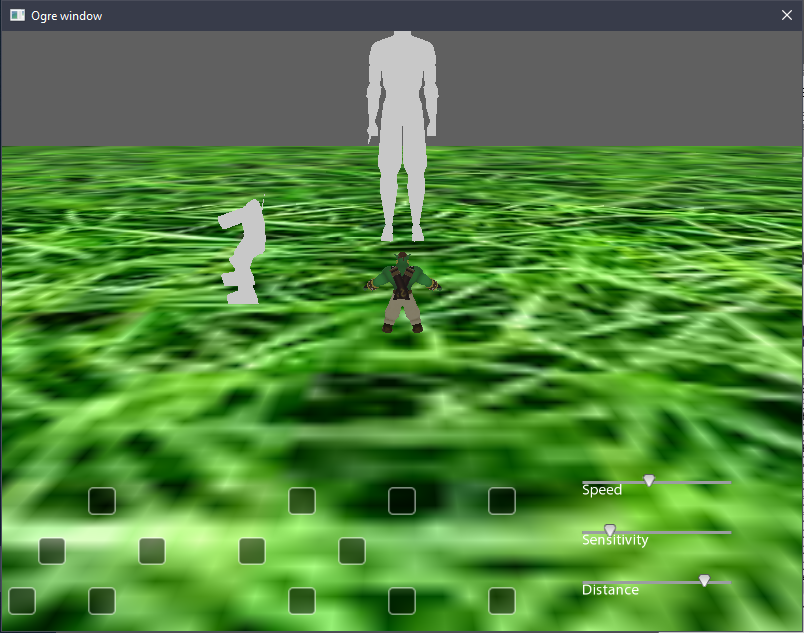
Где:

CollisionBox – создаёт принудительный куб, который заставляет объект быть непроходимым.

VissibleBox – позволит показать куб

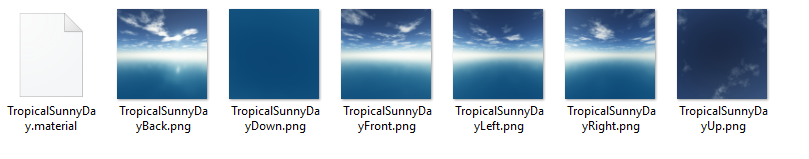
Visible позволяет сделать объект невидимым (если есть принудительная коллизия)

В итоге:



Подключение SkyBox(Небо):

Нам нужен файл .material и 6 изображений:



Где в файле .material прописываем:

material TropicalSunnyDay

{

technique

{

pass

{

lighting off

depth\_write off

texture\_unit

{

cubic\_texture TropicalSunnyDayBack.png TropicalSunnyDayFront.png TropicalSunnyDayLeft.png TropicalSunnyDayRight.png TropicalSunnyDayUP.png TropicalSunnyDayDown.png separateUV

tex\_address\_mode clamp

}

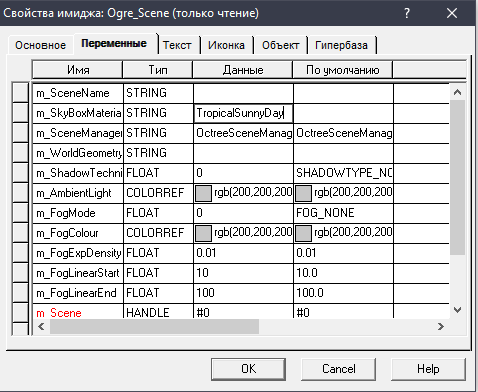
}

}

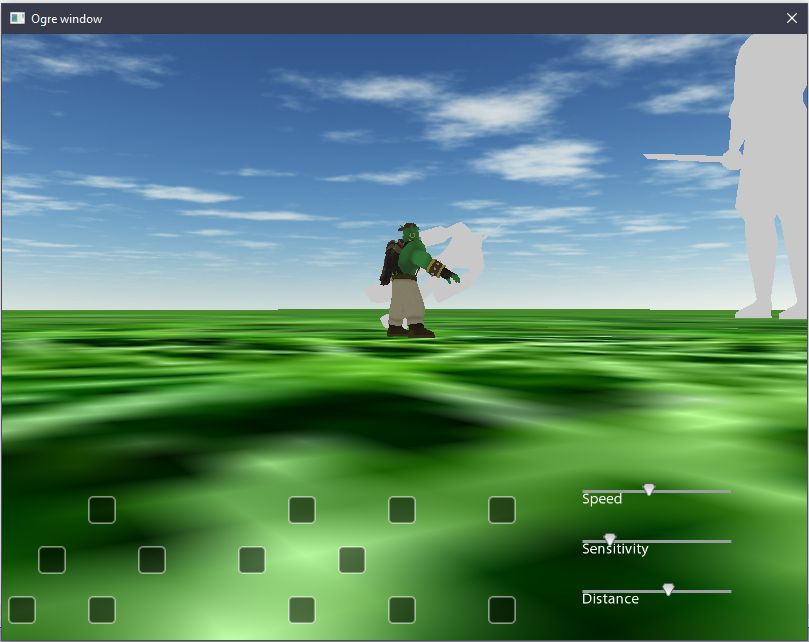
}

В cubic\_texture по порядку прописываем изображения, составляя куб.

Затем в Ogre\_Scene присваиваем имя материала SkyBox:



В итоге получаем небо:



Добавление окон:

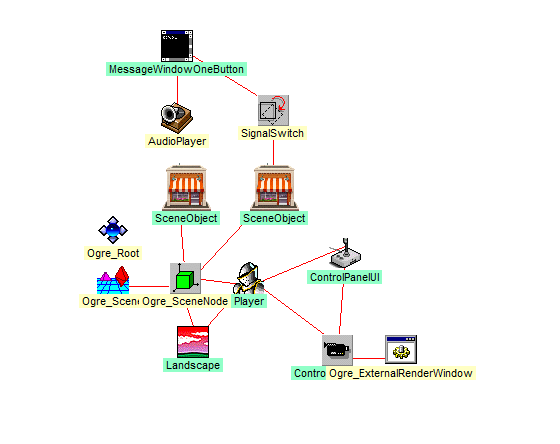
Окно с кнопкой можно получить с помощью класса MessageWindowOneButton

Окно без кнопки с помощью класса

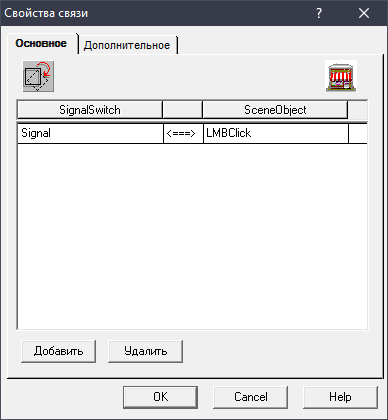
WindowFileText

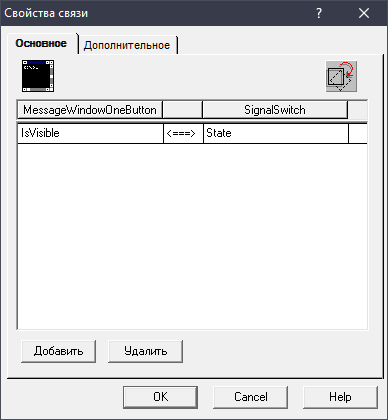
Для звука используем класс AudioPlayer из OgreExtension

Соберем следующую схему:

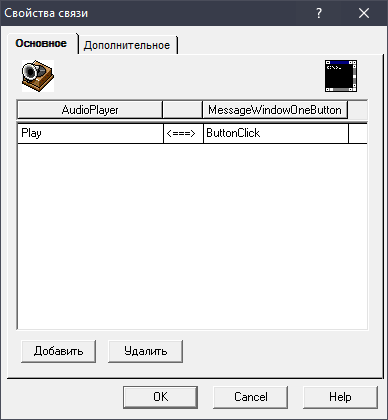


При нажатии на SceneObject кнопками мыши активируются переменный LMBClick, RMBClick, MMBClick -–левая, правя, средняя кнопка мыши. Присоединим к данному свойству имидж SignalSwitch, который будет работать переключателем, где Signal – от SceneObject, State – к окну.

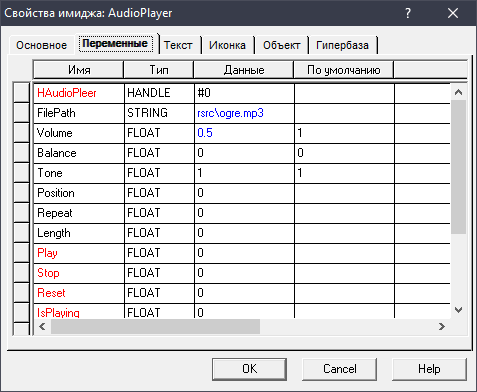




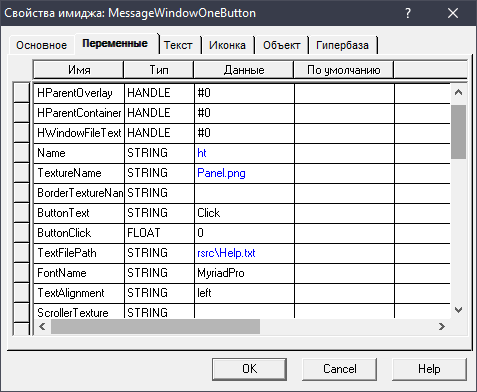
Также соединим аудиоплеер с окном:

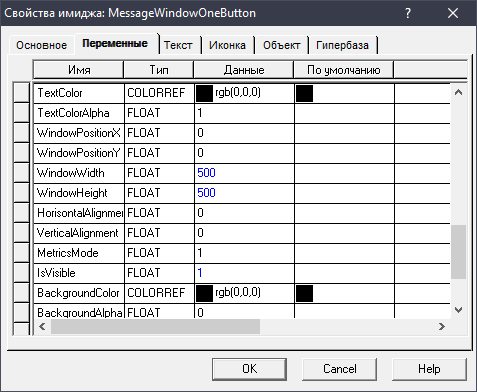


Укажем путь к аудио, можно также указать громкость щвука:

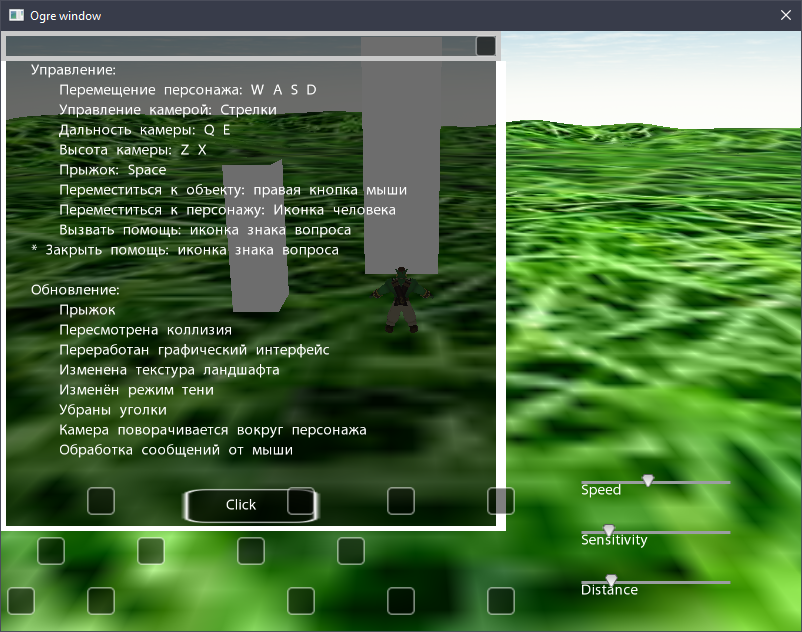


Для окна укажем следующие настройки:





Теперь при нажатии на один из объектов появится окно:



**Источник света**

Типы:

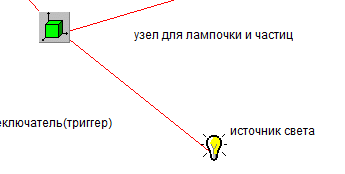
LT\_POINT – точечный

LT\_DIRECTIONAL – направленный

LT\_SPOTLIGHT – конусный

Для использования света необходимы два имиджа:

Ogre\_SceneNode для размещения света и сам имидж света Ogre\_Light



С помощью PowerScale устанавливаем интенсивность.

m\_Diffuse – цвет

m\_Type – тип

**Импорт сцены из 3D Max в Ogre**

1. Строим сцену в 3D Max
2. Экспортируем сцену с помощью плагина, при этом для сцены использовать измерение в пикселях (иначе объекты будут мелкими).
3. Переносим всю структуру папок в ресурсы проекта.
4. Используя функцию Scene\_Load(~Scene, ~SceneName) или имидж SceneLoader подключить сцену к проекту.
5. Для использования объектов сцены используется SceneObject, Player или PhysicObject. При этом свойство AddManually необходимо установить в 0, чтобы объект брался из сцены, а не создавался новый.
6. Обязательно необходимо использовать источник света, иначе объекты будут черными.



**Отображение Ogre3d во внутренних окнах Stratum**:



* Необходим HSPACE
* Имидж Ogre\_RenderWindow

Соединяем ControlledCamera с Ogre\_RenderWindow, а Ogre\_RenderWindow c LGSpace

При это для работы интерфейса, мыши и клавиатуры тоже необходим LGSpace.

Решение: создать свойство HSPACE в главном имидже и использовать его.

**Материалы**

Шаблон файла материалов:

// <-Комментарий

//Название материала

material MaterialName

{

//техника

technique

{

//проход

pass

{

//текстурный блок

texture\_unit

{

//текстура

texture image.png

//масштаб

scale 0.1 0.1

}

}

}

}

**Материал** может состоять из множества техник

**Техника** - это один из способов достижения желаемого эффекта. Вы можете предоставить более одного метода, чтобы обеспечить запасные подходы, когда карта не имеет возможности визуализировать предпочтительный метод, или если вы хотите определить версии материала с более низким уровнем детализации, чтобы сохранить силу рендеринга, когда объекты более отдалены.

Каждая техника может состоять из множества **проходов**, то есть полный рендеринг объекта может быть выполнен несколько раз с различными настройками для получения составных эффектов. **[Ogre](https://ogrecave.github.io/ogre/api/latest/namespace_ogre.html" \o " Этот исходный файл является частью OGRE (движка объектно-ориентированной графики).)** может также разделить определенные вами проходы на множество проходов во время выполнения, если вы определите проход, который использует слишком много текстурных блоков для карты, на которой вы в данный момент работаете (обратите внимание, что он может делать это только в том случае, если вы не используете фрагментную программу ).

Рассмотрим различные варианты настроек материала:

**Сплошной цвет**

material Template/Red

{

technique

{

pass

{

texture\_unit

{

colour\_op\_ex source1 src\_manual src\_current 1 0 0

}

}

}

}

'source1' использует первый источник без изменений.

В этом случае первым источником является следующий аргумент.

«src\_manual» означает - использовать цвет, обозначенный тремя числами в конце.

Числа представляют значения RGB в диапазоне 0 и 1.

'src\_current' - это texture\_unit, который смешивается с 'src\_manual', но в этом случае, поскольку операция должна использовать первый источник без изменений, результирующая текстура - это просто сплошной цвет, обозначаемый «1 0 0».

**Простой затененный цвет**

material Template/Red

{

technique

{

pass

{

lighting on

ambient 0.3 0.1 0.1 1

diffuse 0.8 0.05 0.05 1

emissive 0 0 0 1

}

}

}

material Template/Blue

{

technique

{

pass

{

lighting on

ambient 0.3 0.3 0.3 1

diffuse 0.1 0.2 0.7 1

emissive 0 0 0 1

}

}

}

material Templates/RadioactiveGreen

{

technique

{

pass

{

lighting on

ambient 0.1 0.3 0.1 1

diffuse 0.2 0.2 0.2 1

emissive 0.05 0.8 0.05 1

}

}

}

Все цвета формируются с использованием трех значений от 0 до 1, которые представляют компоненты RGB. Некоторые параметры принимают 4 значения, четвертый - альфа-значение (также между 0 и 1). Значение 1 означает белый или максимальную яркость, 0 означает черный или низкую яркость.   
Ambient (Окружающая среда) определяет основной цвет объекта, без какого-либо освещения к нему. Т.к. цветовая модель является аддитивной, то это означает минимальную яркость объекта (когда он находится в абсолютной тени или не светит на него).

Слишком высокое значение (например, полностью белое (1 1 1)) приведет к тому, что все ваши объекты будут полностью белыми. Установка черного цвета (0 0 0) даст вам полностью черные тени.

Diffuse (Рассеянный) цвет определяет основной цвет вашего материала при наличии источников света. Он определяет, какая часть каждого цветового компонента отражается, определяя, какой цвет будет иметь материал, когда на него светит полный белый свет. Обратите внимание, что окружающий цвет добавляется к результату рассеивания, поэтому, если вы установите значение окружения, вам может потребоваться уменьшить значение рассеивания, чтобы компенсировать это.

Emissive - это количество света или цвет, который испускается от объекта, не нуждаясь в каком-либо свете, и дает эффект, как будто объект освещается своим собственным окружающим светом. Обратите внимание, что излучающий свет не освещает другие объекты в сцене, поэтому, если вам нужен светящийся объект, который освещает остальную часть сцены, вам придется добавить дополнительный свет в сцену. Если вы хотите, чтобы цвет окружающей среды отсутствовал, то вам нужен именно этот параметр, а не окружающий.   
Кроме того, есть «зеркальная» настройка, которая позволяет настроить зеркальную составляющую вашего материала. По умолчанию он отключен (0 0 0 0).   
Чтобы эти параметры (окружающий, рассеянный и излучающий) работали, вам нужно включить «освещение». Также использование 'colour\_op replace' в текстурном блоке вашего материала отключит эти настройки.

**Видимость с обеих сторон**

По умолчанию материал виден только с одной стороны.

Для плоских поверхностей (например, водной поверхности) часто нужна видимость с обеих сторон.

material TransparencyExample

{

technique

{

pass

{

//...

cull\_hardware none

cull\_software none

}

}

}

**Прозрачный цвет**

material Template/Red50

{

technique

{

pass

{

scene\_blend alpha\_blend

depth\_write off

texture\_unit

{

colour\_op\_ex source1 src\_manual src\_current 1 0 0

alpha\_op\_ex source1 src\_manual src\_current 0.5

}

}

}

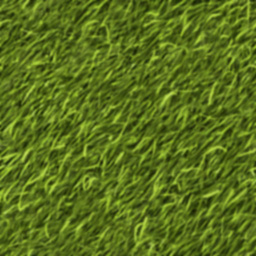
}

Строка 'colour\_op\_ex' объяснена выше. 'Alpha\_op\_ex' имеет ту же структуру аргументов, единственное отличие состоит в том, что значение в конце указывает альфа-значение для использования. Здесь цвет устанавливается на 50% прозрачным. Вы можете поэкспериментировать с различными аргументами для scene\_blend, чтобы придать материалу разные виды прозрачности.

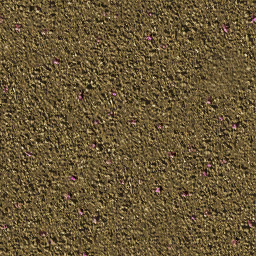
**Простой текстурный переход**

**Требования к материалам**

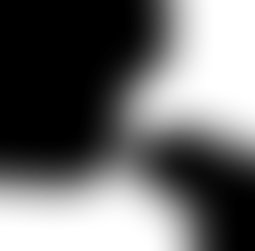
* смешанный файл должен быть в **8-битном** формате (24-битный не работает)



* *Material\_dirt.jpg* - Текстура назначения



* *Material\_alpha\_blend.png* - файл PNG с альфа-маской, которую мы хотим заимствовать для перехода.



Правило смешивания изображения: черная (или цветная) область будет исходной текстурой, а прозрачная область будет целевой текстурой

material Template/texture\_blend

{

technique

{

pass

{

texture\_unit

{

texture Material\_grass.png

}

texture\_unit

{

texture Material\_alpha\_blend.png

colour\_op alpha\_blend

}

texture\_unit

{

texture Material\_dirt.jpg

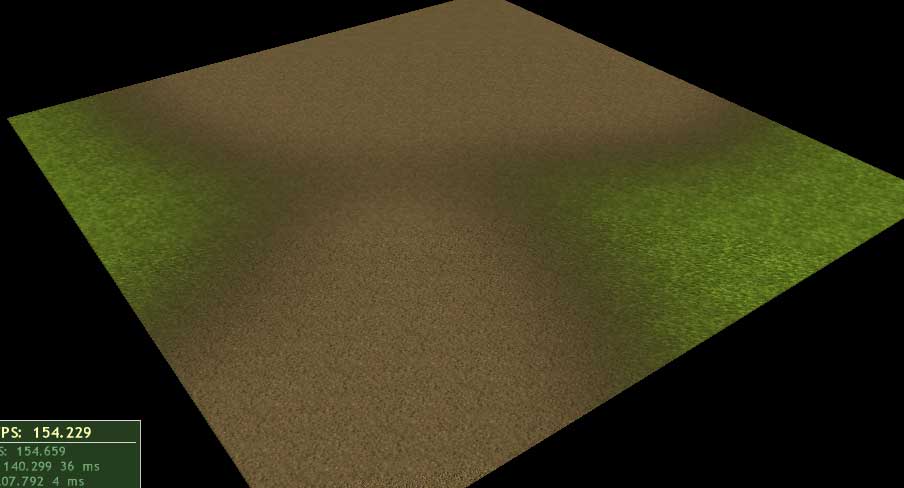
colour\_op\_ex blend\_current\_alpha src\_texture src\_current

}

}

}

}



**Витраж**

Предположим, у вас есть четырехугольник, и вы хотите, чтобы он был частично прозрачным, как витраж.

* image.png - Текстура, которую вы хотите быть прозрачной

material Template/TransparentTexture

{

technique

{

pass

{

lighting off

scene\_blend alpha\_blend

depth\_write off

texture\_unit

{

texture image.png

alpha\_op\_ex source1 src\_manual src\_texture 0.5

}

}

}

}

Подробнее:

<https://ogrecave.github.io/ogre/api/latest/_material-_scripts.html#SEC23>

**Частицы**

Создадим файл с частицами:

WaterDrop.particle

//Дождь

particle\_system WaterDrop

{

//Материал

material WaterDrop

//ширина, высота частицы

particle\_width 0.2

particle\_height 0.2

//Убираем огр. рамку

cull\_each true

//квота

quota 5000

//Тип спрайта – точечный щит

billboard\_type point

//Эммитер типа элмпсоида

emitter Ellipsoid

{

//высота, ширина площади

height 1000

width 1000

inner\_height 0.9

inner\_width 0.9

//частота эмиссии

emission\_rate 500

//Направление

direction 1, -1, 0

//Скорость

velocity 50

}

}

И файла с материалом:

material WaterDrop

{

technique

{

pass

{

lighting on

scene\_blend alpha\_blend

depth\_write off

diffuse vertexcolour

texture\_unit

{

texture WaterDrop.png

tex\_address\_mode clamp

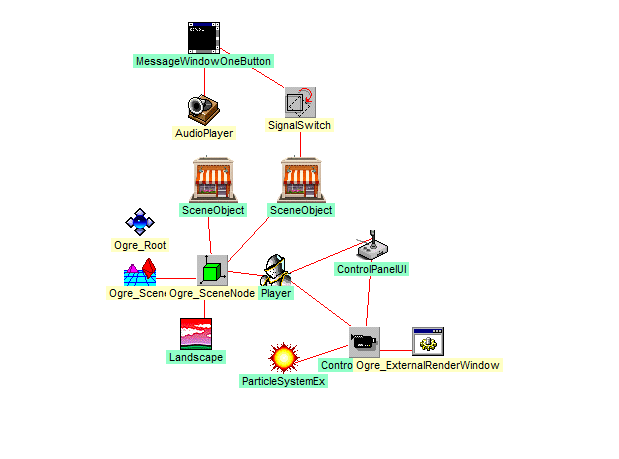
}

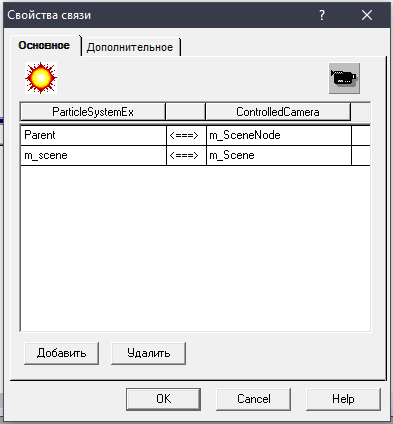
}

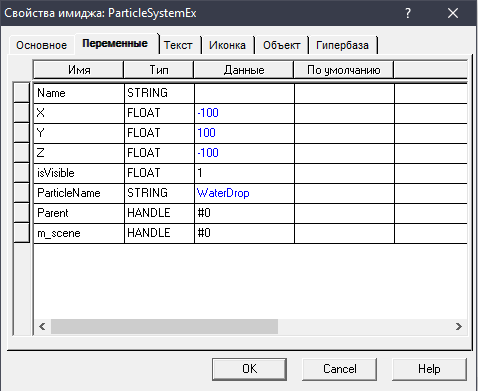
}

}

Создадим такую схему:







Здесь указали название системы частиц и её расположение.

Теперь дождь всегда будет над камерой

