**The Version 1**

This version is now alpha and don’t includes all the following features that will be included in the version 1:

* Complete Audio system management
* Fist physics integration
* Complete shaders development toolkit
  + Create your custom materials for scene nodes (ex: Normal Mapping, water, etc.)
  + Create your custom filters (ex: Sepia filter, Screen Water filter, etc.)
  + Customize your filters using a LUA script including tools
* Complete particles systems management
* Complete scene management
* Complete plugins system management
  + Plugins are dynamic libraries developed by users using the editor’s API to make their editor more powerful. You can also share your plugins.
  + Create your own monitor: A monitor is a plugin to render the scene. You can create a monitor to use the Oculus Rift, etc. The editor includes a monitor names “GenericMonitor” that is the default monitor.
  + Create your own editor plugin toolkit: You can add functionalities to the editor by creating you own toolkit like a new window for nodes edition, a plugin to integrate the “Terrain Painting” toolkit, etc.
  + Create your own scene nodes: You can create your own scene node like a water surface, or your own terrain scene node for example.
  + Create your own animators: You can create your own animators like “Rotation animator” or animators to animate characters for example if you have a preferred technic.
  + Crate your own loaders: You can create your own mesh loaders, or image loaders and then set if the editor uses this loader or not.
* Complete Data management
* Complete Core functions to make work easier
* About the renders:
  + Create and manage your scene nodes
  + Create and manage lights
    - Configure shadows
    - Configure color
    - Configure type of light
    - Etc.
* Complete API to develop your video games that use the editor
* Complete video player integration, rendered in the scene or for cinematic
* Compatible Window, Linux & Mac OS X
* About LUA scripts
  + Scene implementation
    - Scene Node implementation
    - Scene functions implementation
  + Driver implementation
  + Editor relative implementation

**Customize Your Filters**

Once you added a filter, you can edit its code and its callback. The callback is a LUA Script you’ll use to set filter’s shader constants. **Cf. Demo Project.**

Metatable **Filter**.

* setPixelSharderConstantFloat(name, value)
* setPixelShaderConstantVector2D(name, value)
* setPixelSharderConstantVector3D(name, value)
* setPixelShaderConstantColor(name, value)
* setPixelShaderConstantColorf(name, value)
* setPixelShaderConstantMatrix4(name, value)

**Vector2D** is composed of x and y values

**Vector3D** is composed of x, y and z values

**Color** is composed of r, g, b and a values where

* r: red (0-255)
* g: green (0-255)
* b: blue (0-255)
* a: alpha (0-255)

**Colorf** is composed of r, g, b and a values where

* r: red (0.0-1.0)
* g: green (0.0-1.0)
* b: blue (0.0-1.0)
* a: alpha (0.0-1.0)

Metatable **Driver**.

* setTexture(path) : set the texture used by the filter by giving the path
* setRttTexture(name) : set the render target used by the filter by giving the name of this one
* getLightPosition(indice) : returns a Vector3D of the light number “indice”

Metatable **Utils**.

* getCurrentTime() : returns the current time
* getScreenCoordinatesFrom3DPosition(position) : returns a Vector2D that represents the position on the screen of the Vector3D “position”.

Metatable Matrix4.

* TO DO

**Customize your materials (shaders)**

Once you added a material shader, you can customize its constants using a simple language, and also a LUA script. The simple language is here to set easily constants. The LUA script makes constants more complex and more interesting.

(LUA scripts are not yes implemented).

Set a vertex shader constant or a pixel shader constant:

- Floats and Integers:

* v-p - type\_of\_the\_constant – name - value(s). Examples :
  + vfloat bumpDelta 12.9: set a float named “bumpDelta” to the vertex shader with the value “12.9”
  + vint myTexture 0: set an integer named “myTexture” to the vertex shader with the value “0”
  + pfloat myFloat 1.0: set a float named “myFloat” to the pixel shader with the value “1.0”

- Matrixes:

* v-p-matrix name matrix1 matrix2 matrix3 makeInverse Examples:
  + vmatrix4 worldViewProj world[0] view[1] proj[0] makeInverse
    - world returns the current World Matrix
    - view returns the current View Matrix
    - proj returns the current Projection Matrix
    - Parameter [] makes the matrix inverted
      * 0 – Not inverted
      * 1 – Set inverted
    - The last parameter is to set inverse of the concatenation result
      * 0 – Do nothing
      * makeInverse – Inverse the concatenation
  + vmatrix4 world world[0] 0 0 0
  + vmatrix4 worldInverse: 2 ways
    - vmatrix4 worldInverse world[1] 0 0 0
    - vmatrix4 worldInverse world[0] 0 0 makeInverse
  + vmatrix4 projViewWorldInv proj[0] view[0] world[0] makeInverse
  + pmatrix4 viewProj 0 view[0] proj[0] 0
  + etc.

- Manipulate time

* set a vertex or a pixel constant using v,p-constant\_type – constant name - + ctime – division,currentTime . Example:
  + pfloat myFloat + ctime 1000: set a float named “myFloat” to the pixel shader with the current time value divided per 1000.
  + vfloat myFloat + ctime realTime: set a float named “myFloat” to the vertex shader with the current time value (same as if the current time is divided per 1)

**User Interface**

Cf. the screenshots named “User Interface.png” in the same folder

Cf. UserInterface.pdf

You’ll find the “Main Window” that is the “quick build” part of this editor. It automatically assigns nodes to the core data in function of its type:

* “Terrains” are static objects like a factory or a castle and it is the main environment. Terrains can also be height maps products where terrain painting is still in development
* “Trees” are optimized scene nodes to represent vegetation. Maybe useless today because vegetation is still in development and began a few week ago
* “Objects” are animated objects, like characters, or everything else animated
* “Lights” will contain all the lights of your scene. You can edit shadow maps, colors, etc. using this tab
* “Water surfaces” is the tab containing all your water surfaces. Now it’s an hard coded water surface but is going to be customized via scene node plugins
* “Particles Systems” is now WIP

The scene graph contains all collections of main window and an “Others” section. The “Others” section is the free scene management part of the editor. Still at a “To Do” state, you’ll be able to manage your scene graph at the Version 1.

Now, the following combinations are:

* CTRL+E: Edit the selected Node quickly
* Hold Control to manipulate the current camera
* CTRL+SHIFT+E: Edit the selected node materials
* SHIFT+A: Edit animators of the selected node (It is WIP and it’s a test version for managing animators of scene nodes)
* CTRL+A: On an object, it will open a window to edit easily existing frames of your animated object. To enter bones edition simply click on the context menu “Edit” and then “Edit Bones”
* When moving a scene node using crosses, press “Space” and you’ll obtain a free position on axis X and Z + Y using the mouse’s wheel.
* CTRL+S: Save the project, now it saves the project at the editor’s executable root directory
* CTRL+O: Open an existing project
* SHIFT+Left Click: Select the selected object on the 3D context
* Right Click:
  + On the main window
  + On the scene graph