

Open Source Based 3D Scene Editing Tool for Game Developing

Valdir Salgueiro
valdir_salgueiro@yahoo.com.br
Universidade de Pernambuco

Abstract

The creation of games and simulations involves several steps, such as modeling and animation, graphic arts, audio composition, etc. It is necessary to develop tools to facilitate the implementation of these steps and control the entire creation process. This paper proposes the development of a scene editor based on free software that allows editing and viewing a 3D scene in real time. The editor is open source, can be easily modified to integrate new modules and modify existing modules, being based on libraries that have non-restrictive license to use in commercial applications.

Keywords: 3D scene editing, games, game editing tools

1. Introduction

The game development process and 3D multimedia applications requires several steps, where many engines are used during the game production [Deloura 2000][Rollings 1999][Zerbst 2004].

Known as game engine, game framework or just engine, the game engine is the mechanism that controls the game reaction according to user input. The game engine implementation involves many computational aspects, like the choice of an appropriate programming language based in its easy use, effectiveness and portability; the development of specific algorithms, services for scene management and rendering, character control and world control, window application management, audio management, support to math routines, classification and data, communication mechanisms, synchronization, user interface model, among others [Luiz 2000].

Engines automate the sound creation process, music, models, scenes, etc. OGRE [Streeting 2005] is a graphic rendering engine and supports a great infrastructure for exhibition and 3D images rendering. The developer has to choose other necessary engines like physics, sound, input, and others that will ease the project conclusion. That's why it has been largely adopted, because the OGRE engine follows the UNIX philosophy: "do one thing, but do it well" [Raymond 2003], leaving the developer which other tools that will be used and how they will be integrated.

However, the developer's community needs one tool to put all his artistic content and make the game on it. A tool that permits the user to automate many steps that are common in games such as terrain creation, entity positioning, waypoint creation, etc., where they can do the adjustments visually with the help of

intuitive graphical interface and with closed engines quality like Unreal Engine[Epic 1998]. This tool should be able to abstract and to allow the developer view the entire scene, as well as the game entities inside it before starting developing game logic and minimizing total development time of the application. It also should be free, not free as in "free beer", but free in the sense where users can adapt it to and specific need or even help in the development and maintenance. This article shows the result of the scene editor development focused to game development.

The article has been divided in five sections. The second section discusses the theoretical referential used in building the editor and article writing and presents some of the tools and technologies used in the gaming industry. The third section provides an analysis of some major open source rendering engines. The fourth section describes the organization of the codes and some of the requirements that have been raised for construction of the tool. The fifth section presents an overview of the scene editor, while the sixth section presents the conclusions. Finally, in the seventh section brings the references.

2. Theoretical Referential

To develop the tool was necessary to choose an engine or API for rendering models, terrain, shaders and effects that can be used in it. The construction of the engine of a game passes through various stages of choice of architecture and APIs that will be used in development. The industry has lately chosen to rebuild the wheel, starting the design from a graphics API, this choice is mostly limited by the target audience you want to be hit with the game. In recent years many of the games has been directed to the public that owns the Windows operating system, so the industry has been using DirectX as the rendering API, however in some cases there are engines that are built to run on more than one platform, and offer a choice of APIs for rendering in accordance with the operating system or hardware available on your computer.

Some companies hold the title of building engines, and their engines have evolved significantly with each release. As an example we have the Source Engine, Cry Engine, and Unreal Engine, they also allow your engine to be licensed, but the license price is too high for small and even for some medium studios. While this strategy is interesting for a company that can finance the costs of a large engine, most studios do not have sufficient resources to fund such a project, being submissive to the restrictive licensing and price charged by large corporations. Most of the time the

license does not permit modification of the engine to better adapt the game to be developed, which makes it difficult to integrate with internally developed tools.

As access to these commercial engines is restricted to some studios, the user base is very small which leads less chance of error correction and improvements that can be made in future versions of an engine with that type of business model.

The main obstacle however is the fact that each company has to rebuild the wheel, when you want to develop your own engine without relying on other solutions available in the industry. This ends up generating a huge duplication of code and functionality that could be combined in a single library, or in smaller libraries.

An alternative for those companies who cannot afford the cost of a commercial engine, but still need the resources to make a game with state of the art technology, is to use engines for each task, such as rendering engines for display of models, shaders, etc., audio engines, physics, and other engines that are sold or distributed separately and can be assembled to build a larger engine to be used in game development.

3. Rendering Engine Choice

After performing a search of the main engines in use today, the conclusion was that the engines that meet most requirements were OGRE and Irrlicht. We will make a brief comparison between these two engines, and explain why OGRE was chosen as the graphics rendering engine. The main requirements were:

- Be multiplatform.
- Support for OpenGL or DirectX.
- Have active community.
- Good support for shaders.
- Most permissive license.
- Reasonable titles quantity released.

It was made a little comparison of both engines:

3.1. Irrlicht Positive Points

- Very good loading BSP and indoor scenes.
- You can load many different models of formats such as .3 ds, obj, mesh, collada etc.
- Easy to use interface.
- Support for rudimentary physics, however, you probably still need to use a physics engine like ODE [Smith 2001] or Newton [Jerez 2002].
- Small executable and DLL small.
- Format scene by promising and editor IrrEdit.
- Support for older hardware, such as video cards that are no longer being marketed and for operating systems: Windows 98 Windows ME Windows NT 4 Windows 2000 Windows XP Windows XP64, Vista Windows,

Windows CE, Linux, OSX and Sun Solaris / SPARC.

3.2. Irrlicht Negative Points

- Irrlicht is slower to load models.
- Irrlicht not always benefits of newer video cards.
- Support for shader is not fully implemented.
- Less SDKs (Software Development Kit) than OGRE

3.3. OGRE Positive Points

- Libraries for Release and Debug. Good for error checking. There is also an optional memory leaker manager.
- The latest versions of OGRE support threads, which dramatically increases the rendering performance on computers with processors of more than one core.
- It has its own mesh format to optimize performance.
- Scripts material, shader, and overlay. You do not need to recompile your code to simply change some configurations.
- System based on plugins and DLLs. It's easy to create/modify/add scene handlers, rendering plugins, and managers of special effects or the core of the OGRE engine without having to recompile all the engine. It also offers flexibility.
- Optimized to render the latest graphics cards available.
- Large user community broadly participatory

3.4. OGRE Negative Points

- The BSP scene manager is outdated.
- OGRE does not work on old video card.
- OGRE based applications tend to be larger.
- OGRE supports only the format mesh. Not all users like it.

Finally, the OGRE engine was chosen because of the familiarity of the object-oriented interface, its large community contributing many improvements and plugins, high level of games developed using it, a good amount of advanced developers where many of them work in the industry and due to it being modular. OGRE rendering engine has several features such as shadows, particle systems, animation, etc., already implemented, requiring only an understanding of the engine to set the scene with the desired resources. By having a big source code, the engine can overwhelm novice programmers, but its highly object-oriented interface is familiar to most game programmers. The next chapter will make an explanation of the organization of code, install and use the scene editor.

3.5. Development Milestones

It was identified the following activities to finish the tool successfully:

- Conduct a survey of scene editors in the industry and the gaming community;
- List what features are needed in a scene editor.
- Study of the graphics rendering engine OGRE3D;
- Implementation of the features listed;
- Implementation of test cases to validate functionality;
- Receive feedback from the user community and improve the tool.

4. Code Organization

The code is structured based on the MVC architecture, which allows modification of modules without interfering with other editor features; enabling the reuse of code in other projects and optimizing recompilation time only one module of the project. We defined the following conventions for classes and divisions of the editor:

- Classes of presentation, these class files are prefixed with W, for example, the class WLevelEditor, manages the main screen editor.
- Classes of instruments are classes that take care of user input and translate it to the current instrument. Example, the move instrument moves an entity along the rendering panel according to user input.
- Managing classes, manages an entity or an aspect of the editor. For example, XML handlers, input managers.
- Classes have bases EGO_ prefix, eg ego_entity, ego_light, etc.. are the most basic editor entities, such as light, sound, water, etc.

4.1 Classes

The editor source code is divided into several classes and some interfaces. The idea is that each class implements a specific function, making the system as modular as possible.

- **QueryManager** manages the selection of objects as is done in the editor and how the gizmos are applied to them. Currently the selection of objects is made at the polygon level, making the selection more accurate than the default selection that is based on the object bounding box.

- **InputManager** provides an interface that can extend the classes if they need to process user input. This class makes a broadcast input entered by the user for all objects registered in it.
- **Ego_Render** initializes the OGRE and its subsystems. Creates the panels and controls the rendering user input in them.
- **Ego_Camera** controls the movement of the camera.
- **EditorFrameListener** is invoked for each rendered frame. It delegates responsibilities to the active instrument.
- **InstrumentManager** records and manages all the instruments available in the editor. Only one instrument can be active at a time.
- **SimpleInstrument** is an interface which all instruments must implement.
- **Ego_Factory** records all XML parsers available in the editor, and also handles the creation and destruction of all objects in the editor.
- **Ego_ResourceManager** manages all resources as models, materials, scripts, etc.
- **Ego_Scene_XML** provides an interface for creating new XML parsers. Also takes care of loading all registered interpreters in class Factory when a map is loaded or saved.

5. Scene Editor Overview

During the development of the editor, there was a caution with the usability of the interface as well as minimum requirements and ease of installation of the editor. The scene editor is directed to any user interested in game development and should not require prior knowledge of that user to use it. Figure 1 shows the main screen of the scene editor:

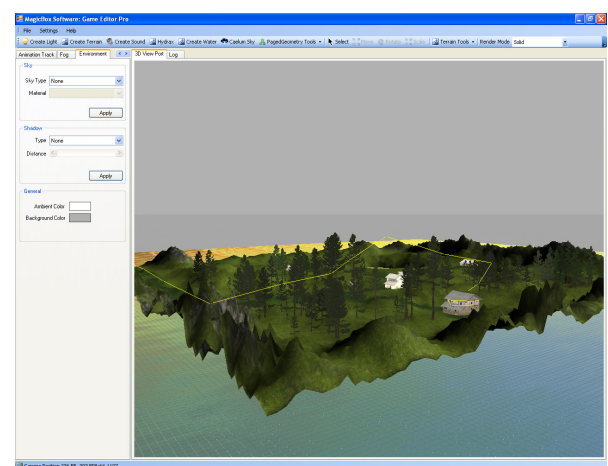


Figure 1. Editor Main Screen

Movement and basic actions can be using the following controls:

- **Camera Movement:** W: Forward, S: Backward, A: Left, D: Right,

- **Rotate Camera:** Mouse right button.
- **Select object:** Mouse left button.

5.1. Minimal Requirement and Installation

The most common installation format is by extracting compressed file containing the entire Editor Data. Make sure you have the latest Visual Studio redistributable and the latest version of DirectX.

These are the minimum requirements to run the editor:

- Windows 2000.
- Processor clocked at 2GHZ.
- 1GB memory
- Video card with Pixel Shader 2.0, with at least 256MB video memory.

If necessary, you can run the editor using the OpenGL renderer through the batch file *opengl.bat*.

6. Final Considerations

The development of this tool allowed learning various aspects of game programming and editors who often are not part of the menu of a university computer course, and practice the knowledge learned during university.

We believe that opening the editor source code will be an increase in user base resulting in the creation of new functionalities and innovative uses not yet anticipated. The growing participation of engines such as OGRE has pointed to a promising future for smaller developers, students and researchers who want to use the best technology in their applications. They will have access to the source code of these libraries extending them and creating a large repository of engines and functions of public use.

The scene editor is essential in building a multimedia application. This tool will allow an increase in the number of developers who could not create their applications before for lack of a tool to assist in their development process. Because of the ease of use and visualization of the scene at the time of creation, there should be an increase in the number of applications completed and a lower rate of cancellation of projects. The scene editor allows applications to possess a commercial state of the art in technology.

Here are some improvements that can be done in the scene editor:

- The editor may be able to create geometries from simple models such as spheres, squares, etc.; these geometries are generated from set functions as union, intersection, difference and so on.

- The editor can allow entities to be positioned in relation to others and in relation to the grid.
- Entities can be grouped into a common node.
- It should be possible to select more than one entity.
- Must be able to undo one or more transactions.
- It should be possible to hide, show or group entities in certain sections.
- Add physical objects.
- Allow to program objects through logical blocks.
- Panel editing shaders and materials.

References

- ADLER, M. 2004. Zlib License. http://www.gzip.org/zlib/zlib_license.html.
- DELOURA, MARK. 2000. 3D GAME ENGINE PROGRAMMING. 1ª ED. CHARLES RIVER MEDIA.
- EPIC. 1998. Unreal Engine. <http://www.unrealtechnology.com>.
- FREE SOFTWARE FOUNDATION. 2007. Lesser General Public License. <http://www.gnu.org/copyleft/lesser.html>.
- JEREZ, J. 2000. Newton Game Dynamics. <http://www.newtondynamics.com>.
- LUIZ, A. 2000. Jogos por computador - Histórico, Relevância Tecnológica e Mercadológica, TENDÊNCIAS e Técnicas de Implementação. Jornada de Atualização em Informática. PUCPR.
- RAYMOND, E. S. 2003. The Art of Unix Programming. 1ª Ed. Addison-Wesley Professional.
- ROLLINGS, A. 1999. Game Architecture and Design. 1ª Ed. Coriolis Group Books.
- SMITH, R. (2001) Open Dynamics Engine. <http://www.ode.org>.
- STREETING, S. et AL. OGRE3D. (2003) <http://www.ogre3d.org>.
- ZERBST, S. DUVEL, Oliver. (2004) Game Programming Gems. 1ª Ed. Course Technology PTR.