

Developing video games and virtual environments with the Crystal Space engine

Jorrit Tyberghein, Eric Sunshine, Frank Richter, Mike Gist
Crystal Space team
<http://www.crystalspace3d.org>

Christian Van Brussel
Communications and Remote Sensing
Laboratory
Université catholique de Louvain
Louvain-la-Neuve, Belgium
christian.vanbrussel@uclouvain.be

ABSTRACT

In this paper, we present the Crystal Space framework (CS – [1]), a complete open-source software development kit for the development of modern video games and virtual environments. We describe the various components of the framework, their interaction with 3D design tools, and how it is possible to use them to create applications with complex virtual worlds. We present more closely the available features and the level of abstraction introduced in the CS SDK and the Crystal Entity Layer (CEL). We also present CELStart, a runtime package environment on top of CEL.

Keywords

Game engine, software development kit, virtual reality, video games, 3D rendering

INTRODUCTION

Nowadays, in accordance with the increasing presence of video games and virtual reality-based software in everyday computer applications and entertainment, there has been a growing need for a greater number of realistic virtual worlds with ever increasing complexity. The size, the level of detail, the realism of the scenes, as well as the need for immersive interaction have all been increased, in turn forcing developers to invent ever more complex solutions to address the issues that are created by such requirements.

In recent years this complexity has lead to a situation where, contrary to the early days of the video games, it is no longer possible to develop a competitive 3d video game with a small team, e.g. the typical trio of one coder, one graphical artist and one musical artist. To address the increasing complexity and quantity of the technical functionalities and artistic creations needed to simulate virtual worlds, the size of the team needed to develop a game has become bigger and bigger, and the budgets involved in the production of modern commercial games

has rapidly caught up with those of Hollywood's big productions.

It has been difficult for independent and low-/mid-budget games to compete with such a level of complexity, and as a result their productions have been ghettoized in to simpler games, e.g. in 2D instead of 3D. For example, the quantity, quality and diversity of games made with Flash technology attests to the willingness of the community to create such applications from the moment the available technology made it possible to concentrate mainly on the content and logic of the game, instead of on technical aspects of the simulation.

In these last few years there has been a flourishing range of middleware and complete frameworks attempting to address this problem, solving part of (or as much as possible of) the technical needs of modern games and virtual reality applications. There are both open-source and commercial libraries and tools for e.g. 3D design and rendering, 3D sounds, physics, animation, artificial intelligence and even complete frameworks integrating most of these functionalities in one single comprehensive software development kit.

Crystal Space is one such complete framework; we will present it in the rest of this paper.

BACKGROUND AND RELATED WORK

One of the first types of middleware for virtual reality appeared with video games such as id Software's Doom and Quake ([2]) in the early and mid 1990's. Part of the huge success of these games came from the fact that they were shipped with tools and mechanisms allowing anyone to modify and extend the content of the game. Soon, many teams of fans formed across the Web and started the creation of new variants of the initial games. The concept of 'mods' (for 'modifications') was born, a phenomenon which had great influence on the future evolution of the game industry. These 'modifiable' games played the role of sandboxes for the imagination of the community, allowing many new concepts to emerge from all these contributions. The noticeable success of mods such as 'Counter Strike' and 'Team Fortress', both of which were later adopted by Valve

*LEAVE BLANK THE LAST 2.5 cm (1") OF THE LEFT
COLUMN ON THE FIRST PAGE FOR THE
COPYRIGHT NOTICE.*

Software and converted into high-quality commercial products, testify the importance of the phenomenon.

Even before that, companies such as id Software (and later on Epic Games with the Unreal license [3]) had started to license the software technology of their games (their 'game engines') so that other commercial companies could develop new games on top of the technology already provided in the game engines. 'Half-Life' for instance, the first game produced by Valve Software, was using a modified version of the Quake engine, whereas this same company now licenses its home built engine 'Source' ([4]), one of the main solutions for game engines today. Besides this new commercial market and along with the emergence of the open-source movement, new open-source projects addressing that same task were created, examples of which are Crystal Space, Ogre 3D ([5]) and the Irrlicht Engine ([6]).

More complex solutions appeared afterwards, such as the CryEngine ([7]) of Crytek, featuring what they called a "What You See Is What You Play" editor (WYSIWYP interfaces). Such editors enable the artist to directly visualize the virtual environment concurrently to its design, providing a new interactive way to build 3D worlds. The editor also contains many advanced features allowing manipulation of the environment from a high semantic level, such as the road builder which can create new roads in a few mouse clicks and automatically adapt the landscape, the flora and any objects along the road. Microsoft XNA ([8]) is another example of a complete framework for the development of video games.

Finally, other companies and open-source projects focused on more specific problems, such as the skeletal animation (Euphoria), the Artificial Intelligence (SpirOps and PathEngine)), network management (RakNet and GNE), and physical simulation (NVIDIA PhysX, Havok Physics, Bullet).

THE CRYSTAL SPACE ENGINE

Overview

Contrary to other open source 3D engines such as Ogre 3D or the Irrlicht Engine, CS is not only a rendering engine, i.e. a system concentrating on the display of 3D scenes, but encompasses many functionalities covering a wide range of the technical/middleware needs of modern virtual reality applications. In this sense, it is closer to a complete game engine (such as the commercial Unreal Engine and Source SDK's) than its open-source counterparts.

Crystal Space was founded in 1999 as a personal project hosted on the SourceForge open source software development web site. The project rapidly received keen interest from the open-source community as it was one of the first open projects on 3D real-time rendering. The statistics attest of the success of the project: on July 2000 the SDK was downloaded 15.000 times, and the web site received a record of 1.300.000 visits on March 2005. More

than 200 contributors have so far participated to the development of CS and the framework has been used in many independent games, as well as in some academic research projects and some commercial games, and it has been a testbed for many experimental features that were later used in other projects, such as the Verse network protocol funded by the EU Sixth Framework Programme ([9]). CS is also more or less closely related to many other open-source communities, such as the teams of Blender ([10]) and PlaneShift ([11]).

The CS project is a typical mid-scale open-source project, based on spontaneous and voluntary contributions instead of on a planned design with financial funding. The work is coordinated through open discussion, under the monitoring of the administrators of the project. Due to the nature of the participations to the development process, there has been no real long-term or global planning of development other than guidance from the main administrators and contributors. Special care is therefore needed in order to keep a clean and coherent global architecture of the system. However, one quality of CS is the relatively good general design of its architecture.

For the last four years CS has been a mentoring organization in the Google Summer of Code program. This program is designed to introduce student developers to the Open Source community and to get them contributing to one of the many highly successful projects that exist. Through this program CS has shown to be a good reference aid in learning the field of computer graphics.

General architecture and workflow

CS is licensed under the GNU Lesser General Public License. It is written mainly in C++, runs on GNU/Linux, MS Windows, and Mac OS/X, and supports a wide range of compilers.

CS encompasses two main components, the CS SDK, providing the low-level functionalities such as rendering, sounds and physics, and the Crystal Entity Layer, providing more advanced features and a higher level of abstraction for manipulating the objects of the virtual environment.

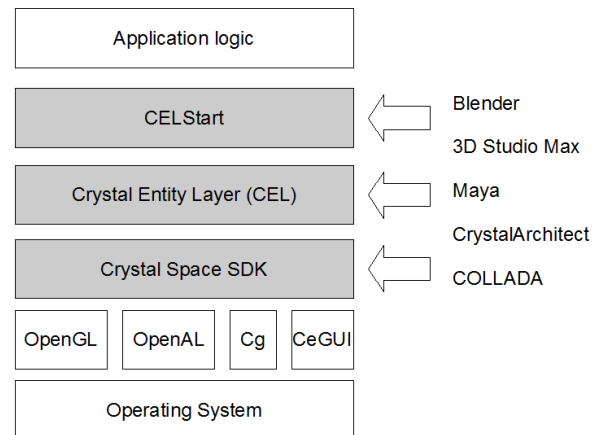


Figure 1 General architecture of the Crystal Space framework.

The general architecture of the whole framework is presented in figure 1. The CS SDK is above the operating system, and uses some few external libraries such as OpenGL, OpenAL, Cg and CeGUI. CEL is on top of the CS SDK, while the logic of the user applications is defined on top of CEL (or directly above CS according to their design choices).

Above CEL sits CELStart; a runtime package environment that allows the developers to distribute 3D applications in a rather innovative and elegant way. CELStart consists of a generic launcher application for 3D software comprised of a single self-contained package, holding the entire assets and logic of the 3D application. The logic of the application is defined through portable scripts written in Python or in a dedicated XML compliant description language. As the data files and scripts are self-contained and portable, the CELStart launcher has only to be installed once on a system. A new application then needs to only distribute a single data package. This allows for an easy distribution, installation and management of application packages.

The assets, as well as the descriptions of the virtual scenes, and the events and behaviors of the objects in the scenes, can be described in CS's XML compliant description language. They are imported into CS and CEL from 3D design tools such as Blender (thanks to the related open-source project blender2crystal), 3D Studio Max, and Maya. CrystalArchitect, another related open-source project, is a design environment with a particular emphasis on CS which allows the direct visualization of the 3D environment concurrently to its design (WYSIWYP interface). There are also some importers for file formats such as COLLADA.

The Crystal Space SDK

One main quality of CS is its modularity, achieved through the Shared Class Facility (SCF), a component-based system based on the COM model. It separates the API of the components from their implementations, which are accessible in the form of shared libraries. The system allows the dynamic loading of the plugins, providing the ability to switch or select plugins at runtime or during development. One main benefit is the structuring of the code that is imposed by this system, forcing the decomposition of the system in independent and interchangeable components and allowing an easy integration of new features. This also helps to address problems such as incompatibilities on different platforms, and makes it possible to extend it with other communication mechanisms for SCF, for example inter-process communication through pipes or CORBA.

There have been some questions asked about the performance penalty of such a modular system, in a field where every bit of extra performance that can be gained is highly desired. However the general consensus among

developers of the project is that the learning and flexibility advantages of this design outweighs the negatives of any performance loss, especially in a world where hardware performance is increasing at such a rapid rate.

The CS SDK contains common features of modern high quality 3D rendering engines, such as shaders and post-processing effects, dynamic lighting, HDR rendering, view frustum culling, advanced skeletal animation, particle systems, collision detection, pseudo-instancing, peripheral management and support for many data file formats.

It has good support for big and/or external environments with a dedicated heightmap-based terrain mesh, a foliage system for the automatic addition of objects such as flora and rocks, impostors and Level Of Detail (LOD) management (allowing the display of a great amount of objects in the same view), and the decomposition of the environments into sectors connected by portals, for effects such as LOD management, streamed loading or space warping.

In addition to the functionalities dedicated to 3D rendering the CS SDK has many other features that are commonly needed when developing 3D applications, which are also already integrated with the other objects of the SDK. The Virtual File System abstracts the access to the data files, whatever they are on the file system of the operating system, or within ZIP packages. There is also support for 3D sounds (through OpenAL), physical simulation (Bullet and ODE), graphical user interfaces (CeGUI), multi-threaded loading, text localization, font server, and debugging tools.

Finally, CS has also bindings for other programming languages through SWIG. It has support for Python, Java, and Perl, and some experiments have been made for Lua and C#.

The Crystal Entity Layer

The main addition of CEL is an entity system which abstracts all technical aspects of the management of the objects, letting the designer concentrate on the content of the virtual environment. Objects are defined as entities, on which property classes defining a facet or aspect of what the objects are and how they behave are attached. An event system enables generic communication between property classes. There is a wide range of property classes already available, covering many of the needs of modern 3D applications. These can also be combined to create complex entities and behaviors. Main property classes are for avatars and Non-Player Character management, camera management, scene and world management, binding of peripheral events, selection and grabbing of objects, advanced physics (e.g. fluids, explosions, wheeled vehicles, gradual destruction of objects), artificial intelligence (behavior trees, path finding, steering behaviors, neural networks and genetic algorithms).

As an example of the combination of property classes, the entity of a user avatar would typically have an 'input' property class to redirect the peripheral events to the entity, a 'mesh' property class for the 3D model of the avatar and its animations, a 'movement' property class for the synchronization of the user actions with the mesh, its position, its animations, and the collisions with the environment, a 'select' property class to enable the avatar to select and grab objects near him, an 'inventory' property class to manage the objects that are grabbed, and finally a 'camera' property class to have the camera automatically follow the avatar.

The level of abstraction introduced by CEL provides an elegant way to manipulate all the objects of a virtual environment. Some tools benefit from this, such as the persistence layer used for saving and loading world states, and the experimental networking layer achieving the synchronization of data among the nodes of a multi-player environment.

The logic of the application is defined through the list of entities and their property class. However the user still has to define how these entities are updated, and how they behave exactly. This can be achieved through C++, Python, or a dedicated XML compliant description language. The quest manager tool also helps by providing the infrastructure for user interaction and for the triggering of events.

CELStart is actually just a small piece of software on top of CEL, providing the functionalities for managing the application package, accessing the XML description of the application and its entities, loading and setting up all entities and property classes, and updating them thanks to the XML and Python scripts incorporated in the application package.

CONCLUSION AND FUTURE WORKS

In this paper, we have presented the Crystal Space framework and its main parts, the CS SDK, CEL, CELStart, and their related software tools and projects. This "Crystal Space umbrella" forms a complete open-source middleware solution covering most of the needs for the development of modern video games and virtual reality environments.

At current time, there are no real open-source counterparts to CS, in the sense that there are really few other complete game engine projects, and that none of them provide so many modern features. Ogre 3D for instance, probably the most used open-source project about real-time 3D, is concentrating only on the rendering problem and do not address such a wide range of functionalities. Even with the large set of external plugins which are available for Ogre 3D, the CS SDK alone stands already out with features such

as advanced skeletal animation, LOD management and streamed loading, and with the fact that all these features are already well integrated together, thanks to the SCF component-based system. In addition to that, the levels of abstraction and the features provided by CEL and CELStart make CS a unique framework and an excellent choice for academic, independent and low-/mid-budget video games and virtual environments, while big-budget projects would still prefer the high-level of quality and the guaranteed assistance provided by commercial products.

However, an advantage of Ogre 3D over CS is the size of its community, something which is important for open-source projects. To face this problem, the next improvements that are planned for CS will begin with a better public image and communication, something which has always been a bit neglected by the contributors to the project. It will start with a new graphical layout and navigation for the website. Other improvements are needed, for instance a more sustained release cycle, as well as the release of binary packages and the development of more attractive and explicative demonstration applications.

New features are also planned, in order to match the evolution of the new technical needs. The CS SDK has recently received the addition of a new system for advanced skeletal animation, many features for which have already been added (e.g. ragdoll and inverse kinematics) and many others are planned (e.g. facial animation and parametric motion graphs). Other domains that are worked on in the CS SDK are advanced lighting techniques such as radiosity and deferred shading, more featured post-processing and environmental effects, and better support for import tools for 3D Studio Max and Maya. For CEL, future improvements are needed in artificial intelligence with e.g. more advanced features for the behavior trees and the path finding system, as well as in networking support.

REFERENCES

1. <http://www.crystalspace3d.org>
2. <http://www.idsoftware.com/business/technology>
3. <http://www.udk.com>
4. <http://source.valvesoftware.com>
5. <http://www.ogre3d.org>
6. <http://irrlicht.sourceforge.net>
7. <http://mycryengine.com>
8. <http://creators.xna.com>
9. <http://verse.blender.org/overview/>
10. <http://www.blender.org>
11. <http://www.planeshift.it>