

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 engine (CS – [1]), an 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. The presentation is accompanied by an overview of the game engine middleware and compares CS to the main other related projects.

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 the domain of video games for instance, the size of the team needed to develop a game has become bigger and bigger in order to be able to address the increasing complexity and quantity of the technical functionalities and artistic creations needed. Because of this, the budgets involved in the production of modern commercial games have 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 the 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. In the rest of this paper, we will present an overview of these software development kits, and then we will present more closely one of them, the Crystal Space engine, and compare it to the main other related projects.

CONTEXT AND RELATED PROJECTS

One of the first types of middleware for virtual reality appeared with video games such as id Software's Doom and Quake 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 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) 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', 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 ([2]) and the Irrlicht Engine.

More complex solutions appeared afterwards, such as the CryEngine 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 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).

All these software solutions cover a wide range of the technical needs for the developing of modern virtual environments. However, it has to be noted that a huge amount of work will most probably still be needed for the creation of the assets of the environment, a process which is currently taking a great part of the man-work needed for a project. In light of this observation, we can probably predict that the developing of new advanced tools assisting the artists in the creation of the assets will be the next challenge to be addressed.

THE CRYSTAL SPACE ENGINE

Overview

Contrary to almost all 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 a complete game engine encompassing as much as possible of the functionalities needed for modern video games and applications using virtual reality.

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

([3]). CS is also more or less closely related to many other open-source communities, such as the teams of Blender and PlaneShift.

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 has therefore been needed in order to keep a clean and coherent global architecture of the system.

General architecture and workflow

CS is licensed under the GNU Lesser General Public License. It is written mainly in C++ using very few non-standard extensions, allowing it to run on a wide range of POSIX platforms such as GNU/Linux, most BSD's, Mac OS/X, as well as on all 32- and 64-bit MS Windows. It has also been recently used successfully on an ARM architecture. Finally, CS supports a wide range of compilers and IDE such as GCC, Visual C++ 8 and 9, MinGW/MSYS and Cygwin.

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.

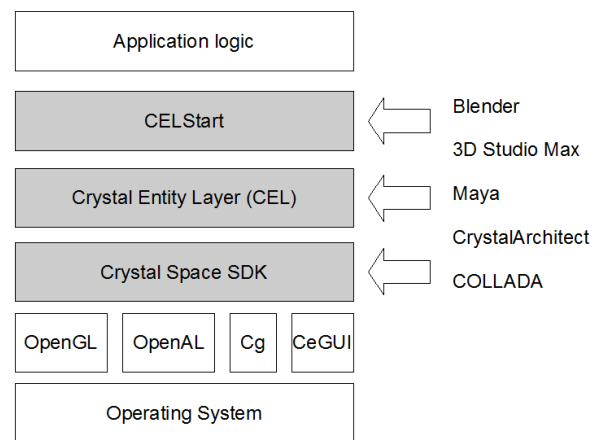


Figure 1 General architecture of the Crystal Space framework.

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, Cg, OpenAL 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, allowing 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 3D scenes file formats such as COLLADA.

To develop an application using CS, one has to choose at first if he will use or not CEL, because a later integration of it will be difficult. As we will see, CEL is versatile and provides a solution for the developing of applications needing almost no programming. With CEL, it is possible to design all or a great part of the application from graphical editors such as Blender and CrystalArchitect, with no or a minimum of coding. However, the level of abstraction and the functionalities provided may not be judged interesting regarding the architectural restrictions imposed. In this case the developing can be made atop of the CS SDK, but it will need the coding of more functionalities (but with more freedom).

As CS covers a wide range of the technical needs of the application, most of the coding can still be concentrated on the logic of the application, i.e. the various concepts and objects of the environments and how they interact. A good start for a new project is to look at other demo and test applications such as CrystalCore or YoFrankie!

The Crystal Space SDK

An important feature of the CS SDK 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.

Another important characteristic of the CS SDK is the wide range of functionalities which are already integrated inside the SDK, providing the base tools to develop any kind of virtual environments, and not being restricted to a single type of application or game. This is reinforced by the fact that these functionalities are organized through SCF and can be selected according to the needs.

Notable features of the CS SDK are a very descent modern 3D rendering engine with shaders, dynamic lighting and post-processing effects, many types of different meshes, and many mechanisms for the management of Level Of Details (LOD) and big and/or external environments. It has good support for physical simulation thanks to the Bullet and ODE libraries, and a strong skeletal animation system using animation blending trees, with support for ragdoll and inverse kinematics. It has also many other features such as abstraction of the operating system and of the hardware peripherals, text localization, font server, integrated graphical user interfaces with CeGUI and wxWidgets, and 3D sounds with OpenAL.

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 precisely. 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 package.

The biggest restriction introduced when using CEL is the fact that most of the concepts and objects of the virtual environment will have to be expressed in terms of entities and property classes in order to be able to interact with CEL. This would allow benefiting of all the generalization features of CEL, but may not be appreciated. That is the reason why one has to choose early in the development process whether he will use CEL or only the CS SDK.

FUTURE WORK

The first improvements which are needed in CS concern its accessibility to the public, with 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 and many others are planned (e.g. facial animation, parametric motion graphs and avatar personalization). 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, hardware occlusion culling, advanced LOD management, 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.

CONCLUSION AND USER PERSPECTIVE

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.

When having to choose for a game or a rendering engine, one has to consider its real needs. For a big-budget production, the best choices are commercial solutions such as Unity, the Source engine, the Unreal engine, or the CryEngine, providing both a high-level of quality and guaranteed user assistance. For academic, independent and low-/mid-budget productions, the open-source counterparts are nowadays more and more interesting. If one simply needs a 3D rendering engine, then CS, Ogre 3D and Panda 3D are good choices. If the user wants to work specifically on the rendering techniques, then he may orient himself to other projects such as GTP ([4]) or G3D ([5]). For some specific video game type, there are some engines well suited, such as ORTS ([6]) and the Spring engine for real-time strategy games. For other less specific or more substantial projects, then a general game engine such as CS is a really good choice, other solution being the Blender Game Engine and the GameKit ([7]).

REFERENCES

1. <http://www.crystalspace3d.org>
2. Gregory Junker, Pro OGRE 3D Programming, Apress, 2006.
3. Uni-Verse project, Deliverable D 1.2, Sixth Framework Programme, 2005.
4. Mateu Sbert, Jordi Palau, GameTools: Advanced Tools for Developing Highly Realistic Computer Games IVth ITRA World Conference, Alicante, July 2005.
5. Morgan McGuire, The G3D Graphics Engine: Advanced language features for simplicity and safety in a graphics API, C/C++ Users Journal, 2004.
6. M. Buro and T. Furtak, On the Development of a Free RTS Game Engine, GameOn'NA Conference, Montreal 2005.
7. <http://code.google.com/p/gamekit/>