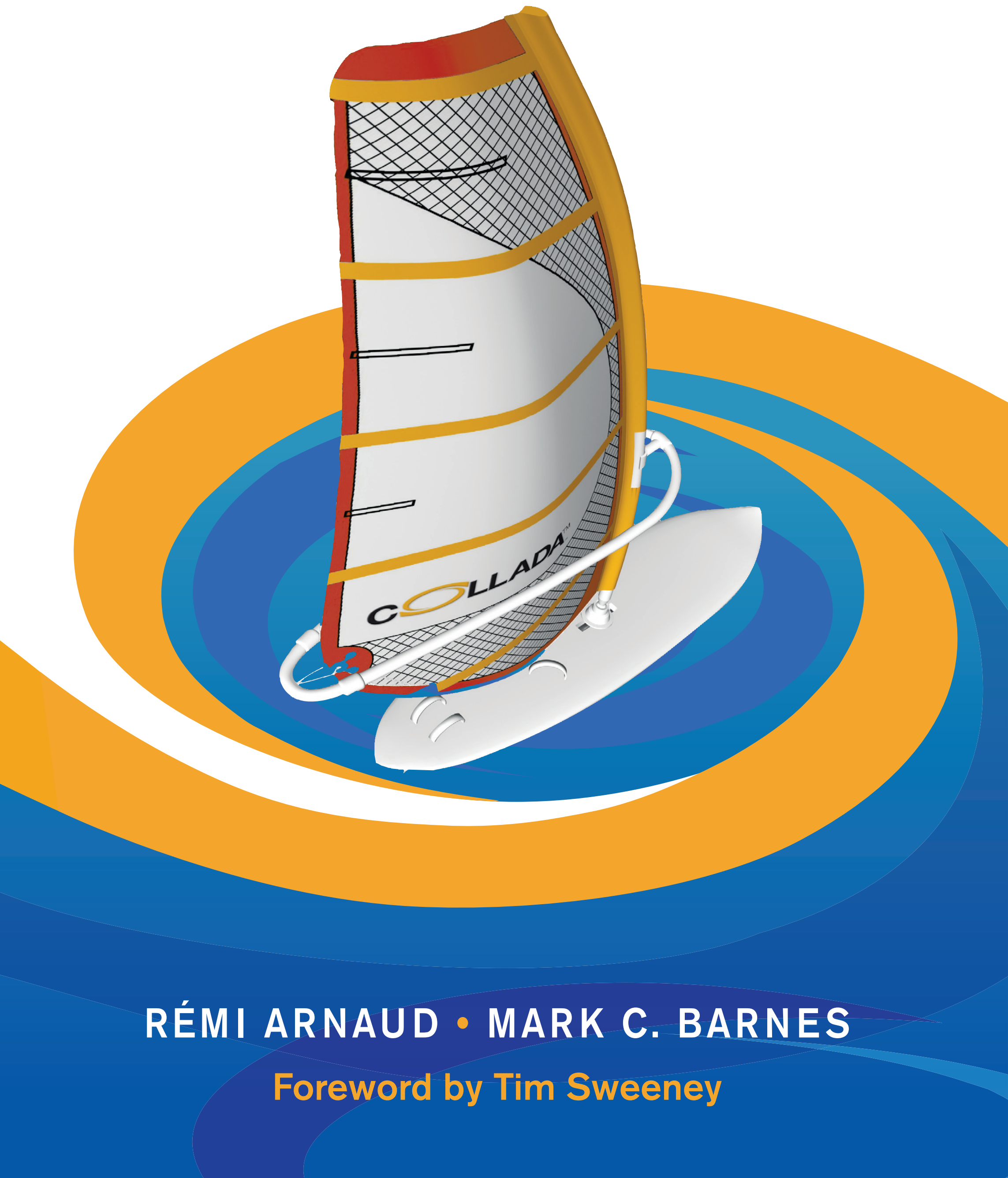
A Khronos Group Open Standard

COLLADA™

SAILING THE GULF OF 3D DIGITAL CONTENT CREATION



RÉMI ARNAUD • MARK C. BARNES

Foreword by Tim Sweeney

COLLADA

COLLADA

Sailing the Gulf of 3D Digital Content Creation

Rémi Arnaud
Mark C. Barnes

A K Peters, Ltd.
Wellesley, Massachusetts

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

Version Date: 20150121

International Standard Book Number-13: 978-1-4398-6579-8 (eBook - PDF)

Contents

Foreword

by Tim Sweeney

In 1963, Ivan Sutherland wrote Sketchpad [57], the world's first real-time computer graphics application. In that breakthrough effort, he defined the basic techniques for structuring scene data that are still present in today's 3D applications.

In one way, though, Sutherland had it easy. He didn't have to grapple with the problem that has frustrated every subsequent developer: interchange, the process of transporting 3D data between applications.

For more than 40 years, developers have implemented interchange using ad-hoc data formats and proprietary interchange techniques. Every new 3D modeling program defines its own proprietary scene data format and also implements partial support for importing data from other proprietary formats.

For example, in developing the Unreal Engine, my company has written importers for five scene file formats, and export plug-ins for the three major 3D applications. Each required significant development and testing effort. What's worse, every time we wanted to add a new feature, we had to update five import modules and three export plug-ins.

This is one of those classic software problems that makes a developer scream: "There must be a better way!"

Finally, there is a better way: **COLLADA**.

Developed in a cooperative effort between the industry's leading developers of applications, games, and platforms, COLLADA is the industry's first standard interchange format for digital content. It is an XML-based file format supporting the transfer of common types of 3D data between applications: 3D models, vertices, polygons, textures, shaders, transformations, lights, cameras, and much more. Just as importantly, it is an extensible format that will grow to support increasingly sophisticated 3D features as they evolve.

Given that 3D interchange is a decades-old problem, one might wonder why a solution is emerging only now. To answer this, we can look to three industry trends.

- First, since the advent of the Web and the XML standard in the 1990s, there has been a growing realization across the computing industry that standard interchange formats benefit all users and application vendors. XML has provided a unifying force because it eliminates the need for debate about the syntactic nuances of a file format, enabling industry groups to instead focus on defining the high-level content and data types in an XML Schema.
- Second, there has been a blossoming of innovative new 3D tools in recent years. Where previously an artist would work in one all-compassing modeling package such as 3D Studio Max, Maya, or Softimage, it is now common to use many specialized tools every day. For example, artists working on 3D games frequently use ZBrush for crafting organic objects, SketchUp for architectural modeling, and Modo for polygon modeling—as well as Max, Maya, and Softimage for scene composition. This workflow entails frequently moving data between applications.
- Finally, the growth of 3D game development means that hundreds of thousands of developers now must frequently move data between modeling applications and real-time game engines. The growing sophistication of the datasets has further stressed the need for an interchange format.

And now, thanks to COLLADA—and to the many people and companies whose efforts made it possible—industry-standard 3D interchange is now a reality, and developers and users of 3D software are already starting to benefit.

For example, my team is replacing the Unreal Engine's (see Plates XI and XII) many ad-hoc file importing modules and our proprietary content-export plug-ins with a single, unified COLLADA pipeline. From now on, when we implement a new feature or data format, we will only have to implement it once, using industry-standard XML tools, instead of multiple times in different code paths. At long last, there is a better way!

More importantly, many new kinds of 3D applications and tools will develop around the COLLADA standard over the coming years, and COLLADA will evolve as developers further the state of the art. It is wonderful to see that, more than 40 years after Sketchpad and the invention of real-time computer graphics, the industry is continuing to leap forward with such innovations.

Tim Sweeney
Founder, Programmer, CEO
Epic Games

Preface

Who Should Read this Book?

COLLADA is an advanced 3D asset description that was created through the collaboration of many partner companies and was adopted as an official industry standard by the Khronos Group in January 2006. It is supported by many commercial and noncommercial tools and is already used by thousands of developers.

COLLADA technology is freely available. It consists of the following:

- the formal specification document written in a specialized computer language representation: the COLLADA schema;
- a human-readable specification document, created from the schema, that describes each element one by one and provides additional guidelines.

This book was created as a guide to the COLLADA 1.4 specification with the goal of providing readers with all the information that will help them understand the concepts, learn how the technology is already implemented by various tools, and provide guidance for using COLLADA in their applications.

In addition, the authors wanted to provide insight into the design of COLLADA so that readers could better understand how the design decisions were made and how this standard may evolve. Since COLLADA covers such a wide spectrum, this information supplements the basic specification material to help developers understand how this technology is intended to be used.

Content developers interested in exchanging data between several tools will find valuable information in this book. Application developers planning to take advantage of COLLADA in their tool chain and tool providers wanting to add COLLADA compatibility will find this a very useful guide.

In addition, this book provides readers with a good review of current asset pipeline technology and can be used as academic material.

Why Is COLLADA Important?

The quantity, complexity, and expected quality of content is growing exponentially with each revision of hardware. Most of the focus so far has been on providing APIs and programming languages to help with the growing complexity and cost of software development, but little effort has been made providing technology to help with content development.

COLLADA's goal is to foster the development of better content tools by standardizing a common intermediate representation that encourages better content quality as well as facilitates the inclusion of many advanced features. A standard, freely available content description technology such as COLLADA liberates the content from proprietary formats, empowering developers to improve the quality and feature set of the content that digital content creation (DCC) and middleware tools are providing.

COLLADA also provides a metric that developers can use to evaluate the best tool for their needs (conformance test) and to validate content against the specification (schema validation).

Being a standard ratified by many companies, COLLADA also ensures the permanence of the availability of the content. It is particularly important in the current context of market consolidation where companies and tools can cease to exist without warning.

The reliance on a particular DCC tool's proprietary format forces developers to store the entire DCC tool and operating system along with the proprietary data for archival purposes. Games created 20 years ago are resurfacing today as classics, not unlike classic motion picture revivals. Game data was a perennial asset then, but the open-source nature of COLLADA and its Unicode data encoding helps to alleviate this problem.

Unfortunately, for vendors who dominate the market, there is a clear business advantage in having an opaque storage format and exposing it only through a limited, proprietary interface, because their customers' data is captured in the proprietary format of the tools they use. Since most of the data is stored in this format, competitors will not have access to all of it, and the cost of changing tools becomes proportional to the amount of content that the developer has already developed. This also makes it impossible for other vendors to provide innovative technologies easily. Even if another vendor added advanced features to their own tools, those features could not and would not be used by most developers until they became available in the main vendor's access interface and storage format. As time passes, customer innovation and creativity is compromised as well because they cannot author content that their DCC tool doesn't support. Luckily there is a counterbalance: since developers have to be able to access the data they need, a tool that would enable them to create content without providing access to it would not be useful for this market. DCC vendors have created application frameworks that enable their customers to plug in custom tools to compensate for the lack of features and to enable content import and export. This has become a large secondary market that diverts time and energy away from the true goals of advancing the features and capabilities of DCC tools. However, the tremendous cost of this market dynamic is becoming too great for the market to bear. A new approach is needed.

Why We Wrote This Book

The idea of writing a book on COLLADA to help with the adoption of the technology has been slowly germinating, but what really triggered this book happened during the Dublin Eurographics conference in 2005. At the end of a presentation about COLLADA, several questioners asked about the availability of additional material in order to use the technology in the professional or academic world.

Alice and Klaus Peters were also attending the conference, where they encouraged one of the authors to work on this project. Their trust, professionalism, knowledge of the business of technical publishing, and respect for the author's work are the main reasons this book has been written.

We hope you have as much fun reading this book as we had writing it.

About the Authors

Rémi Arnaud joined Sony Computer Entertainment US R&D in January 2003 as the graphics architect for the PlayStation®3 graphics API. Rémi has a wide range of experience in designing application interfaces, software tools, and runtimes for real-time graphics applications. He obtained his PhD in real-time image synthesis while working in the R&D department of Thomson Training and Simulation (Paris, France) designing visual systems for custom hardware and high-end workstations. He then moved to the US and worked for Silicon Graphics, where he was in charge of adding high-end features to IRIS Performer, a multiprocessor optimized scene-graph application. During this time, one of the features added through a hardware and software extension was the calligraphic light point capability mandatory for the most advanced civil aircraft training simulators (FAA Level D). The necessity to add this capability to content creation tools brought Rémi to collaborate with Mark Barnes for the first time. Rémi and one of his colleagues, Christopher

Tanner, soon caught the Silicon Valley start-up virus and created Intrinsic Graphics and codesigned the respected Alchemy cross-platform middleware solution for game development.

Mark Barnes joined Sony Computer Entertainment US R&D in July 2003 as a member of the graphics team where he is leading the effort on COLLADA. Mark's experience and knowledge in the field of visual simulation includes database tools, distributed processing, and real-time graphics. Mark was a member of the system software team supporting the Vertical Motion Simulator (VMS) laboratory at NASA Ames Research Center for several years, developing and improving real-time software for graphics, data acquisition, and simulation execution. The VMS lab was also the first customer of an innovative 3D modeling tool called Multigen. Mark eventually accepted a position at Multigen-Paradigm, Inc. (then Software Systems), where he became responsible for the OpenFlight database format and IRIS Performer tools and integration. He was also an engineer on the innovative SmartScene project. During this period of time, Mark established close working relationships with the Performer team at Silicon Graphics (SGI) that was soon to include Rémi Arnaud. Subsequently, Mark was an engineering manager of Muse Communications, Inc., developing advanced virtual environment software for the Internet. He also worked as a staff engineer at IVAST, Inc., developing MPEG-4 client systems.

Acknowledgments

Several contributors have helped the authors to create this book.

Daniel Horowitz from NVIDIA, in addition to his role of chairman of the COLLADA FX sub-working group, has been very generous to contribute the entire content for Chapter 5, "COLLADA Effects."

Christian Laforte from Feeling Software has provided 3 appendices to this book: "COLLADA Plug-In for 3ds Max," "COLLADA Plug-In for Maya," and "COLLADA FX Plug-In for Maya."

Alexandre Jean-Claude from Avid/Softimage has provided the "SOFTIMAGE|XSI 5.1 Plug-In" appendix.

Special thanks to Altova GmbH for putting together a wonderful application called XMLSpy that we used to create many of the illustrations in this book.

The authors are deeply grateful to Sony Computer Entertainment (SCE) for accepting and financing this R&D project. In particular, we extend our thanks to Dominic Mallinson, Director of US R&D, and to Senior Manager Attila Vass, who have given their trust and support to this project and encouraged the authors to create this book in their free time. Ken Kutaragi,

CEO/President of SCE, Masayuki Chatani, Corporate Executive and CTO of SCE, and Teiji Yutaka, Vice President, R&D Division of SCEI, are to be thanked for having accepted the COLLADA strategy and inclusion in the PlayStation®3 software SDK.

The COLLADA team at SCE has provided many hours of hard work to create several revisions of the specification and sample software. Gàbor Nagy has participated in many aspects of the design, chaired the Physics sub-working group, created many demonstrations, and integrated COLLADA in his own modeler, Equinox 3D. Lilli Thompson created the first COLLADA conformance test, relentlessly tracked issues, and created a good deal of COLLADA content. Richard Stenson led the effort of delivering COLLADA to PS3 developers. Andy Lorino edited the COLLADA schema and developed the COLLADA DOM API. Robin Green wrote the first COLLADA FX specification, the largest single part of COLLADA, and chaired the sub-working group. Greg Corson worked on the COLLADA RT and the COLLADA DOM libraries, tracking and fixing many bugs. Three interns also helped with the COLLADA project: Daniel Horn helped implement the first viewer, Fabien Goslin created sample code, and Philippe David created the Refinery asset conditioner.

Many people at Sony Computer Entertainment provided help and support to this project: Lia Adams, Geoff Audi, Michael Budwig, Guy Burdick, David Coombes, Erwin Coumans, Mark Deloura, Nat Duca, Jason Doig, Ellen Finch, Richard Forster, Roy Hashimoto, Alan Heirich, Masatomo Ito, Tatsuya Iwamoto, Vangelis Kokkevis, Antoine Labour, Dmitri Makarov, Bruno Matzdorf, Axel Mamode, Care Michaud-Wideman, Ed Owen, J. Patton, Amy Pilkington, Sébastien Rubens, Aoki Sachiyo, Vlad Stamate, Pip Stuart, Gregg Tavares, Andrew Walker, Yoshinori Washizu, Mike Weiblen, Rob Withey, and Mason Woo.

COLLADA design would not have been possible without the participation of many people from several other companies.

Jeff Yates, now working at Havok, believed in the project since its inception, participated in the design, and created the first 3ds Max plug-in when he was working at Discreet. Jean-Luc Corenthin and his management, Michel Kripalani and Marc Petit, have since managed this project at Autodesk/Discreet.

Gordon Bradley working at Autodesk/Alias participated in the design and wrote the initial plug-in for Maya. Joyce Janczyn, Jerôme Maillot, Michel Besner, Kevin Tureski, and Steven Roselle have provided support for the project.

Alexandre Jean-Claude from Avid/Softimage created the plug-in for XSI and is still working on this project and providing help with the design. Many other people helped at Softimage: Gareth Morgan, Marc Stevens, James Rogers, Alain Laferrière, Simon Inwood, Luc Bolduc, and Takashi Umezawa.

The team at Feeling Software (Christian Laforte, Guillaume Laforte, Zhang Jian, Antoine Azar, Alfred Leung and Misako Matsumoto) provided COLLADA with fantastic, professional-grade plug-ins for 3ds Max and Maya, as well as extensions such as COLLADA FX and COLLADA Physics for Maya and are now working on the COLLADA 1.4 conformance test.

The Emdigo team (Christopher Tanner, Cédric Perthuis, Steve Gleiztmann, and Rory Mather) participated in the design, created the first version of the COLLADA DOM, and took the risk of using COLLADA as the core of their technology for their start-up.

The AGEIA team (Stan Melax, Andy Hess, Emmanuel Marquez, Grady Hannah, John Ratcliff, Mikael Skolones, and Greg Stoner) helped tremendously with the design of COLLADA Physics.

The NVIDIA team (Sébastien Dominé, Ignacio Castano, Daniel Horowitz, and Chris Maughan) really embraced COLLADA for their software tools strategy and is providing COLLADA with great tools and design contributions.

The Khronos Group is doing a wonderful job, particularly Neil Trevett (President), Elizabeth Riegel (marketing), Andrew Riegel (events), and Tony DeYoung (webmaster).

There are many other people involved and supporting COLLADA. Our deepest apologies for any not included here: Farshid Almassizadeh (Electronic Arts), Karthic Bala (Vicarious Vision), Adam Billyard (Criterion Software), Steven Collins (Havok), David Burke (Epic Games), Mark Daly (NVIDIA), Patrick Doane (XLGames), Jerome Durand (EkoSystem), Cass Everitt (NVIDIA), Chris Grimm (ATI), Jason Hoerner (THQ), Chas Inman (NVIDIA), Chris Keogh (Havok), Mark J. Kilgard (NVIDIA), Bill Licea-Kane (ATI), Mikael Lagré (Blender), Kathleen Maher (Peddie Research), Bryan Marshall (Codemasters), Nathan Martz (Double Fine Productions), Jay Moore (GarageGames), Kari Pulli (Nokia), Callan Mclnally (ATI), Tom McReynolds (NVIDIA), Kevin Norman (Maxis/EA), Bruno Patatas (ViaRender Systems), Jon Peddie (Peddie Research), Nicholas Perret (Omegame), Mark Rein (Epic Games), Randi Rost (3Dlabs), Tim Sweeney (Epic Games), Kevin Thacker, Stephen Wilkinson (Nokia), Chris Wynn (NVIDIA), Jenny Zhao (Vicarious Vision/Emdigo).

Rémi Arnaud would like to give a very special thanks to his wife, Cécile, and his two kids, Melody and Nicolas, for enduring many hours alone while he worked on this book, and for their love and support.

Mark Barnes gives a heartfelt thanks to Amy Arreola for her love, compassion, and inspiration.

Trademarks

1 Introduction to COLLADA

Overview

This chapter explains why the COLLADA technology has been developed. It provides a global view, defines the problems addressed by the technology, the main actors, and the goals and perspectives for this new technology. It provides an historic overview and information on how COLLADA is being designed and adopted. The goal is to give the reader an insight into how the design choices are made and how this technology might evolve.

Problem Domain

An interactive application is composed of two major components:

1. the application, which provides information in real time to the user and the means to interact with it;
2. the content, which contains the information through which the application navigates and provides a view to the user.

COLLADA focuses on the domain of interactive applications in the entertainment industry, where the content is three-dimensional and is a game or related interactive application. Therefore, the user of the application will be referred to as the *player*.

The types of information that can be provided to the player depend on the output devices available. Most games use one or several screens to display the visual information, sometimes a system with stereo visualization, and a set of speakers for the audio information. Often, some physical sensation can be rendered, as simple as a vibrating device embedded in the joystick, or as sophisticated as a moving cabin in arcade settings.

The application may output, or *render*, several *sensors* at the same time, often in different places. An *observer* is a term that defines a group of sensors that move together. For example, an observer in a (virtual) car may have at least two visual sensors to represent the out-of-the-windows view (in this case, the view through the windshield) and the rear-mirror view. Several observers, which may be simultaneous users, can be displayed together by the same application, for example, in split-screen games where two or more players are sharing the same screen.

The content must include all data required by all the sensors that the application wants to use. The content can have multiple representations stored, partially sharing some of the elements. For example, if the content represents a landscape, different materials may be needed to represent the four seasons. The different representations of the data are called *scenes.*

Another type of interactive application is a training simulator in which the goal is to teach the user how to behave in real situations. These applications are not games, and the trainees' reactions when they make a deadly mistake in a simulation make this quite clear. COLLADA does not focus on simulation applications, but it could certainly be used in this domain as well [1].

The computer-generated animation movie industry is also very interested in COLLADA. This application is completely scripted, not interactive. That industry's goal is to be able to produce previsualization of scenes that look as final as possible, in a very small amount of time, which is why they are interested in integrating game technology in their production process.

Other types of applications may also profit from COLLADA, but its goal is to concentrate on interactive game applications, not to expand the problem domain. Other applications that require the same type of technologies will indirectly benefit from it.

Separation between Content and Runtime

The first interactive real-time applications rendering three-dimensional graphics required very expensive dedicated hardware and were used mainly in training simulation. Physical separation between content and runtime did not exist in the early applications (such as in the GE Apollo lunar landing trainer) [2]. The content was embedded in the code, or more specifically, some subroutine was coded to render a specific part of the content. Eventually, effort was made to store the embedded content as data arrays, and the code became increasingly generic so it could render all kinds of data.

The next logical step was to separate the data physically from the code. This allowed creating several products with the same application, but with different data. More products were defined by the data itself, and content creation soon became a completely separate task.

The real-time application was then referred to as the *runtime*, and the content for the runtime was stored in the *runtime database*. In the game industry, the runtime is called the *game engine*.

Digital content creation (DCC) tools were created, but the data structures and algorithms used for modeling did not match with the data that can be processed in real time by the application. DCC tools were also used by the movie industry for the production of computer-generated movies in which an advanced rendering engine was attached to the tool to produce a set of still images that compose the frames of the movie.

DCC tools and advanced rendering techniques, such as ray tracing [3] or shader languages such as RenderMan [4], required more advanced concepts than a real-time application could handle. Mathematical descriptions of surfaces such as splines and Bézier surfaces became necessary in the computer-aided design (CAD) market [5].

Interactive applications needed both the advanced modeling techniques and the simpler representation usable in real time. Because of this, compilation techniques, used to create binary executable code from high-level languages, were adapted for the content processing. The database used in the DCC tool was therefore called the *source*. The data compiler takes the source data and creates the runtime data.

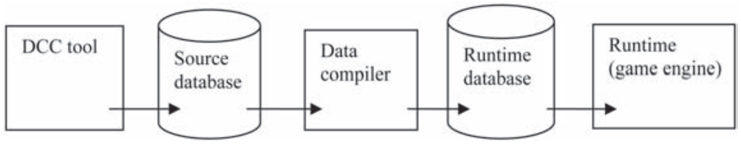


Figure 1.1. Content pipeline synopsis.

The runtime database soon became too large to fit in the memory of the target system. The content had to be sliced and paged in real time, depending on where the observers were. This quite challenging problem sometimes required specific hardware assistance and necessitated a very specific encoding of the content. Specific algorithms had to be developed for terrain paging [6] and texture paging [7].

Because of economic constraints, severe limitations exist in hardware targeted by the game industry. The data must be organized in the most optimal way possible. For performance optimization, many game applications use their own file system and combine the various elements in a single file. Some developers run optimization programs for hours to find the best placement of the elements for better interactivity. The idea is to optimize seek time and place data accordingly. This is very similar to the optimization section of a compiler, with a complexity similar to the NP-complete salesman optimization problem [8].

Another example of paging technology used outside the game industry is the Google Earth application [9]. This application enables the user to look down on the planet from any altitude and render a view based on satellite altimetery and imagery information. It is the result of generalizing the terrain- and image-paging technology developed for high-end simulation applications [10].

Such applications handle only static data and have limited interactivity for the user to move around in the environment. However, most applications, especially in the entertainment industry, require the content to be highly dynamic. For example, animated objects are needed, either objects moving independently in a scene (moving object) or prescripted animations controlled by various parameters, such as a windsock depending on the direction and speed of the wind, or a feature like a bridge that can have different representations depending on whether or not it has been destroyed.

The objects in the runtime database are often organized in a graph, where each branch represents the relative position of the objects or different choices of multiple representations. This data organization is commonly referred to as the *scene graph*, and several runtime technologies have been developed to exploit this [11].

Some early rendering techniques used a particular organization of the scene graph to determine how the objects hide each other in the view, such as the BSP method [12]. Early hardware accelerator performance was greatly affected by the objects sent outside of the camera's field of view, so scene graphs were organized with a container-type relation. All the "children" bounding volumes are enclosed in the "parent" bounding volume; therefore a culling operation can cut entire branches if the parent is not in the field of view.

More complex dynamic behavior is now required for interactive applications. The content needs to evolve following direct interaction with the user. This can be physical interaction with the virtual representation of the user in the application, or it can be indirectly linked to user interaction, such as when a pile of boxes collapses if one of the boxes is removed. Very complex control systems are developed to combine scripted animation, artificial intelligence (AI), physical simulation, and user control.

In other words, the content must be designed for interactivity, not only for the interaction with the user but also for the interactivity between the different elements of the content. The relationship between objects is much more complex, and the scene-graph technology is reaching its limit in capacity, because the graph becomes too complex and overloaded with many interdependent relationships. The dynamic nature of the entertainment application is such that the scene-graph technology is not sufficent to manage all of the content. Instead, hybrid techniques are developed, often targeting a specific game genre. Fortunately, modern rendering hardware performance is less impacted by sending objects outside the field of view, so the main rendering optimization that the scene-graph technology required is no longer an issue.

The process of creating the runtime database from the source database has become more complex and resource-intensive over time. The simple link between the content creation and the runtime is now an entity of its own, called the *content pipeline*. With the need for larger and more interactive content, game developers have to spend substantial resources on this technology.

The Content Pipeline

The content pipeline is composed of the following elements:

- digital content creation (DCC) tools used by artists to create the source data;
- the exporter, a program written for a given DCC tool that permits the content to be extracted from the DCC tool;
- the conditioning pipeline, a set of programs that apply several transformations to the content, such as geometry cleaning and optimizing for fast rendering;
- the runtime database, specifically encoded for a given runtime and often for a given target platform.

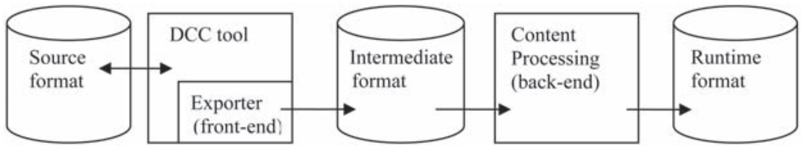


Figure 1.2. The content pipeline.

A novice may ask why it is necessary to write an exporter, since the DCC tool already saves the data in the source database, or why an importer couldn't be created as part of the conditioning pipeline tools. There are, in fact, many difficulties that make these solutions impractical.

The main problem is that the format used by the DCC tool is often proprietary, so it is not possible to write an importer. Even when the format is available, it may be very complex to read and require knowledge of the DCC tool algorithms that are not available to users.

In practice, the exporter is an integral part of the pipeline, since it is already doing some processing on the source data, utilizing the DCC built-in functions to convert internal representation to a more usable format. From the point of view of compiler technology, the exporter is actually the front end of the data compiler, and the data produced by the exporter is the intermediate format.

To enable users to extract the data, DCC tools typically offer a software development kit (SDK) that provides an application programming interface (API) to interact with the internal representations. Some DCC tools provide different APIs, depending on the type of data most often needed by the application. For instance, a game SDK is sometimes provided specifically to help game developers.

DCC tool vendors prefer providing and supporting SDKs rather than publishing the details of their internals and supporting developers tinkering with reading their proprietary files directly. The main reason for DCC tools to do this is that they need to be able to provide new releases. If there were applications depending on a given internal structure, this would make major improvements very difficult. It is thus much easier to hide the internals and provide stability at the SDK level.

Therefore, application developers are forced to create exporters. Even with the relative stability of the SDK, developers must still continuously update their exporters if they want to keep up with new releases of tools. Experience shows that developers often decide to stick to one specific release of a particular DCC tool for a given production, since the cost and risk associated with constantly updating the content pipeline is too intensive. This also affects the development of advanced technology by DCC vendors for game developers who are in the middle of a game-development cycle.

Interestingly, there is also a business advantage since this locks developers into using given vendors and restricts the window of time in which competitors can have a chance at acquiring a new customer. Retooling often happens only when new hardware is introduced, whether it is a new-generation console or a new type of hardware such as a mobile phone. Even then, switching from one DCC tool to another is problematic because of the artists' familiarity with a given DCC tool interface.

Exporter development is not an easy task, but it is forced upon developers. All game developers agree that creating and maintaining an exporter is very time consuming. Nevertheless, they also understand that the content pipeline is a piece of technology they have to master and cannot depend on pieces that are not perfect.

Not only is the resulting game content limited by the game developer's capacity in developing a good content pipeline, but more often, better tools or technologies cannot be introduced because of the lack of flexibility in the design of content pipelines.

COLLADA was created to address this worrisome situation.

Problem Description

To export the data, developers have to design a format in which to export it. This is no simple task, especially because the format must be flexible enough to withstand changes in requirements during the development process, such as introducing new data types.

Once the data is exported, developers still have to write an importer for this data into their content processing facility. Some developers try to solve the problem by having the entire content pipeline contained in the exporter code, so that the output of the export process is directly in the format needed by the runtime. This approach is often problematic since the plug-in interface is not designed to handle such complex applications. It is also the cause of major maintenance problems when the DCC application SDK is evolving and because of the complete lock-in to a given DCC tool and often to a specific version of this tool.

Another approach developers take is to limit the input data to the simplest data, such as geometry, texture mapping, images, and animation curves, and to create a set of integrated tools to create the remaining data, often including the game engine as the means to visualize the data. This proves to be a quite successful approach for developers whose goal is to concentrate on a specific subset of game applications and who can create their content with a relatively small team. Such a tool, if well designed and implemented, provides artists and game designers with a very short feedback loop between content creation and its visualization in the runtime. On the other hand, this approach has several limitations. For example, the edits made in the tool cannot be pushed up the pipeline; therefore, if the input data needs to be modified, all the edits have to be done again. Another drawback is that it is impossible to use external technologies without integrating those directly into the tool itself.

These approaches are in fact contrary to the improvement of the situation, which should be to require an opening up of the tool pipeline to enable developers to use a variety of independent tools, easing the introduction of new technologies, and making possible the adaptation of the content pipeline to be used by larger teams and for all genres of games.

The Zoo

Content pipelines often use more than one intermediate format, since each tool may have its own export format or may not provide an SDK or plug-in facilities for developers to use their own format. In other words, this is a zoo!

The data is exported from the modeler in a given format. An independent tool is used to adjust some properties that the DCC tool does not understand, then another tool is used to gather several assets into a game level, which is then sent to the final optimizer that will create the game-specific runtime format. Each of those tools may very well use independent and incompatible formats since it is difficult to create one format for all. It is so much more convenient for individuals to create their own tools rather than having to collaborate and spend valuable time on making compromises.

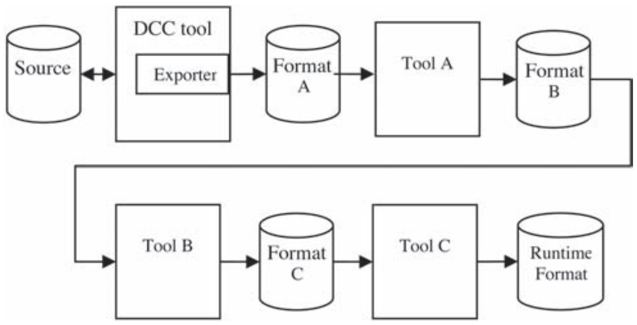


Figure 1.3. A typical zoo.

This process has several limitations.

- The content pipeline is one-way. Therefore, changes cannot be saved back into the source database. When changes need to be done from the source, the data must go through the entire processing of the content pipeline before they can be seen in the level-editor tool. This process takes time, thus impacting productivity.
- The tools are not interchangeable, which often creates situations where data have to be propagated through "back doors" when a change occurs in the process.
- There is no easy way to create shortcuts in the process to enhance productivity.

Artists need to see their work running in the runtime. Ideally, they need to be able to do so without having to go through the entire content pipeline process. Therefore, a separate path, called the *fast-path,* needs to be created in parallel. To maintain the highest possible performance, only the data that is modified will go through the fast-path; the rest of the data will be loaded in the optimized runtime format. During production, it is necessary for the game engine to be capable of loading both the optimized data and the intermediate format.

A Common Intermediate Format

Everything would be much simpler if all the tools in the content pipeline could export and import a well-defined common format. Developers would not need to write and maintain their own exporters, and the data would be available directly to the content pipeline.

COLLADA's goal is to foster the development of a more advanced content pipeline by standardizing a common intermediate representation, encouraging better quality content, and bringing many advanced features to the standard. COLLADA should be the transport mechanism between the various tools in the content pipeline.

But is it possible to establish such a common format?

DCC tools have been developed independently and may have very different ways of describing the data. Some tools have very specific attributes that some developers want to use in their pipeline but which would be impossible to export from other DCC tools. Defining a set of common representations that can be exported by all the tools and making sure that the tool-specific parameters can still be represented is a hard task.

The DCC Vendors

In the entertainment industry, there are three major DCC tools:

- 3ds Max;
- Maya;
- XSI.

All vendors understand the value of a common format, but for a different reason. They are seeking not only an intermediate format but also an interchange format.

The goal of an interchange format is to enable the data to move freely from one tool to another. The main idea is to enable several DCC tools to be used by developers in the same production or to enable developers to switch easily from one main DCC vendor to another DCC vendor. Of course, a DCC vendor who has a much larger market share than the others may not be interested in risking it and may not want a common interchange format to exist, or at least not one that is not under his control.

Recently, Autodesk, who owned 3ds Max, acquired Maya. This consolidation of the market creates a difficult situation for game developers since Autodesk may use its strong position to reduce the interchangeability of data with external tools. At the same time, the need for interoperability between Maya and 3ds Max is growing, unless one of the tools is to be scavenged and