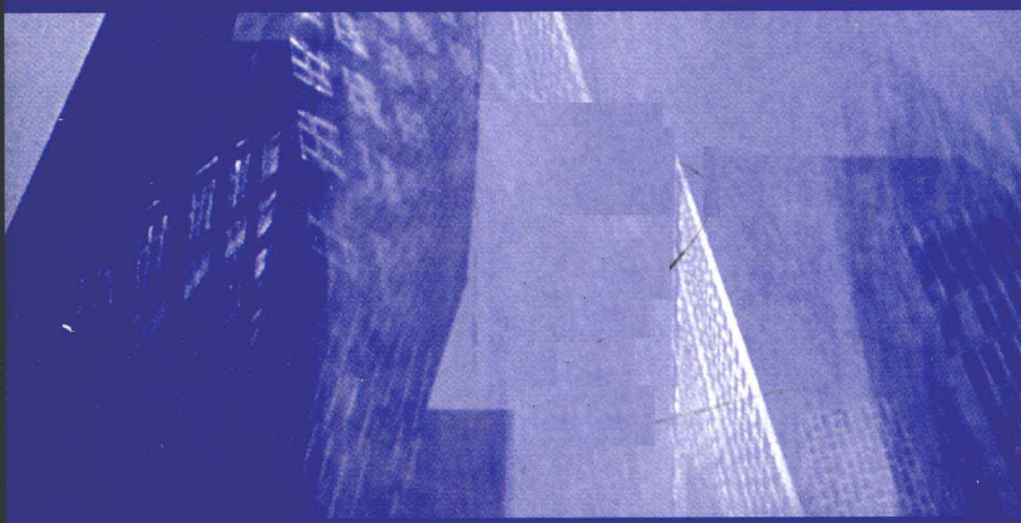
ADVANCED TOPICS IN SCIENCE AND TECHNOLOGY IN CHINA

国家科学技术学术著作出版基金资助出版

Hujun Bao
Wei Hua

Real-Time Graphics Rendering Engine



ZHEJIANG UNIVERSITY PRESS
浙江大学出版社

Springer

国家科学技术学术著作出版基金资助出版

Hujun Bao
Wei Hua

Real-Time Graphics Rendering Engine

With 66 figures, 11 of them in color

ZHEJIANG UNIVERSITY PRESS
浙江大学出版社

Springer

图书在版编目(CIP)数据

实时图形绘制引擎技术 = Real-Time Graphics Rendering Engine : 英文 / 鲍虎军, 华炜著. —杭州 : 浙江大学出版社，2010.12
(中国科技进展丛书)
ISBN 978-7-308-08133-7

Ⅰ. ①实… Ⅱ. ①鲍… ②华… Ⅲ. ①计算机制图—英文 Ⅳ. ①TP391.41

中国版本图书馆CIP数据核字(2010)第227911号

Not for sale outside Mainland of China
此书仅限中国大陆地区销售

实时图形绘制引擎技术
鲍虎军　华炜　著

责任编辑	黄娟琴
封面设计	俞亚彤
出版发行	浙江大学出版社
	网址：http://www.zjupress.com
	Springer-Verlag GmbH
	网址：http://www.springer.com
排　　版	杭州中大图文设计有限公司
印　　刷	浙江印刷集团有限公司
开　　本	710mm×960mm　1/16
印　　张	19.5
字　　数	495
版 印 次	2010年12月第1版　2010年12月第1次印刷
书　　号	ISBN 978-7-308-08133-7 (浙江大学出版社)
	ISBN 978-3-642-18341-6 (Springer-Verlag GmbH)
定　　价	130.00元

版权所有　翻印必究　　印装差错　负责调换
浙江大学出版社发行部邮购电话 (0571)88925591

ADVANCED TOPICS IN SCIENCE AND TECHNOLOGY IN CHINA

ADVANCED TOPICS IN SCIENCE AND TECHNOLOGY IN CHINA

Zhejiang University is one of the leading universities in China. In Advanced Topics in Science and Technology in China, Zhejiang University Press and Springer jointly publish monographs by Chinese scholars and professors, as well as invited authors and editors from abroad who are outstanding experts and scholars in their fields. This series will be of interest to researchers, lecturers, and graduate students alike.

Advanced Topics in Science and Technology in China aims to present the latest and most cutting-edge theories, techniques, and methodologies in various research areas in China. It covers all disciplines in the fields of natural science and technology, including but not limited to, computer science, materials science, life sciences, engineering, environmental sciences, mathematics, and physics.

此为试读,需要完整PDF请访问: www.ertongbook.com

Preface

A real-time graphics rendering engine is a middleware, and plays a fundamental role in various real-time or interactive graphics applications, such as video games, scientific computation visualization systems, CAD systems, flight simulation, etc. There are various rendering engines, but in this book we focus on a 3D real-time photorealistic graphics rendering engine, which takes 3D graphics primitives as the input and generates photorealistic images as the output. Here, the phrase "real-time" indicates that the image is generated online and the rate of generation is fast enough for the image sequence to be looked like a smooth animation. For conciseness, we use the rendering engine to represent a 3D real-time photorealistic graphics rendering engine throughout this book.

As a rendering engine is a middleware, users are mainly application developers. For application developers, a rendering engine is a software development kit. More precisely, a rendering engine consists of a set of reusable modules such as static or dynamic link libraries. By using these libraries, developers can concentrate on the application's business logic, not diverting attention to rather complicated graphics rendering issues, like how to handle textures or how to calculate the shadings of objects. In most cases, a professional rendering engine usually does rendering tasks better than the programs written by application developers who are not computer graphics professionals. Meanwhile, adopting a good rendering engine in application development projects can reduce the development period, since lots of complex work is done by the rendering engine and, consequently, development costs and risks are alleviated.

In this book we are going to reveal the modern rendering engine's architecture and the main techniques used in rendering engines. We hope this book can be good guidance for developers who are interested in building their own rendering engines.

The chapters are arranged in the following way. In Chapter 1, we introduce the main parts of a rendering engine and briefly their functionality. In Chapter 2, basic knowledge related to developing real-time rendering is introduced. This covers the rendering pipeline, the visual appearance and shading and lighting models. Chapter 3 is the main part of this book. It unveils the architecture of the rendering engine through analyzing the Visionix system, the rendering engine developed by

the authors' team. Lots of details about implementation are also presented in Chapter 3. In Chapter 4, a distributed parallel rendering system for a multi-screen display, which is based on Visionix, is introduced.

In Chapters 5 and 6, two particular techniques for real-time rendering that could be integrated into rendering engines are presented. Chapter 5 presents an overview of real-time rendering approaches for a large-scale terrain, and a new approach based on the asymptotic fractional Brownian motion. Chapter 6 presents a variation approach to a computer oriented bounding box tree for solid objects, which is helpful in visibility culling and collision detection.

This book is supported by the National Basic Research Program of China, also called "973" program (Grant Nos. 2002CB312100 and 2009CB320800) and the National Natural Science Foundation of China (Grant No. 60773184). Additionally, several contributors have helped the authors to create this book.

Dr. Hongxin Zhang and Dr. Rui Wang from CAD&CG State Key Lab, Zhejiang University, China, have made key contributions to Chapter 2 "Basics of Real-time Rendering". Ying Tang from Zhejiang Technology University, China, has done lots of work on tailoring contents, translating and polishing this book.

Special thanks to Chongshan Sheng from Ranovae Technologies, Hangzhou, China, as one of the main designers, for providing a lot of design documents and implementation details of the Visionix system, which is a collaboration between Ranovae Technologies and the CAD&CG State Key Lab.

Many people who work, or have ever studied, at the CAD&CG State Key Lab, Zhejiang University, provided help and support for this book: Rui Wang, Huaisheng Zhang, Feng Liu, Ruijian Yang, Guang Hu, Fengming He, Wei Zhang, Gaofeng Xu, Ze Liang, Yifei Zhu, Yaqian Wei, En Li, and Zhi He.

Hujun Bao
Wei Hua
Hangzhou, China
October, 2010

Contents

此为试读,需要完整PDF请访问: www.ertongbook.com

1

Introduction

In this chapter, we are going to introduce the main parts of most rendering engines and, briefly, their functionality. Fig. 1.1 shows a classical structure of a rendering engine. In this graph, the rendering engine is composed of offline toolkits and a runtime support environment. The offline toolkit mainly comprises the tools that export data from the third party modeling software and the tools which perform some pre-operations to the Scene Model. These pre-operations include simplification, visibility pre-computation, lighting pre-computing and data compression. Besides these tools, the more advanced rendering engine includes some special effects generators. The main parts supported in Runtime are Scene Model, Scene Model Management, Scene Graph Traversal and Render. The applications call Scene Model Management and Scene Graph Traversal to run the rendering engine. In the following paragraphs, we will give a brief description of the core parts of the runtime support environment.

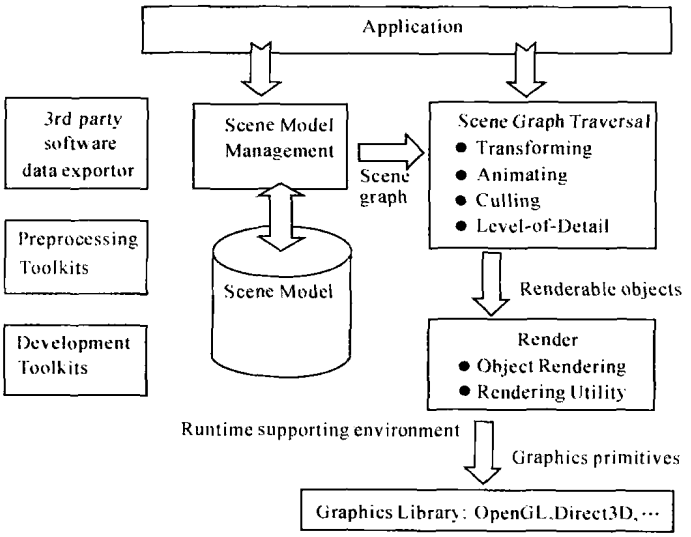


Fig. 1.1 The classical structure of a rendering engine

1.1 Scene Graph Management

Inside a rendering engine, the scene model is the digital depiction of the virtual world in cyberspace. For most rendering engines, the scene model adopts graph data structures, which is called the scene graph. The scene graph is a directed acyclic graph, where nodes represent the entities of the virtual world and arcs represent relations between these entities.

In a scene graph, different kinds of nodes represent different classes of entities. The two most fundamental nodes are renderable objects and light sources. The renderable objects represent the objects that can be displayed on the images produced by rendering engines. The light sources stand for the sources of light, which describe light intensity, emission fashion, position, direction, etc. Given a scene graph, the prime function of rendering engines is to use the light sources to illuminate the renderable objects, and render the renderable objects according to certain viewing parameters.

Undoubtedly, renderable objects are the most important class of entities. In object-oriented architecture, the renderable object is a subclass of the base class entity. To represent various perceptible entities in a virtual world optimally, there is a variety of subclasses of renderable objects. Although these subclasses of renderable objects may look quite different, most of them have two parts in common, geometry and appearance. The geometric part describes the outline, contour, surface or volume of a renderable object. The number of geometric types that can be supported is regarded as an index to measure the performance of a rendering engine. The polygonal mesh or, more precisely, the triangular mesh is the most widely supported geometric representation for all rendering engines with a simple structure. The polygonal mesh can be used to represent most geometric entities and can be easily mapped to graphics hardware. Some more advanced rendering engines adopt a spline surface as the geometric representation to describe finer surfaces. The rendering engine aiming at scientific visualization would support a volumetric dataset. The appearance part describes the optical characteristics of material that constitutes the surface or volume of a renderable object. Many visual effects of renderable objects are dependent on it. Since textures are well supported, on all modern 3D graphics cards the appearance part often uses multiple textures to record various optical properties on surfaces.

The arcs in a scene graph represent the relations between renderable objects. Most rendering engines implement the scene graph by tree structures. In a tree, different types of nodes represent different node relations. The most common relationship is a grouping relationship and the corresponding node is a group node. A group node represents a group of entities and the entities inside the group become the child node of this group node. A grouping relationship is very useful, for it is used to model the hierarchy of the virtual world. In some rendering engines, a group node has a transformation field, which depicts a coordinate transformation for all children in the group's coordinate frame.

Besides a grouping relationship, there is another kind of important relationship

between nodes—reference. The reference here is similar to the reference in C++. The goal of adopting a reference here is to improve the reuse efficiency and decrease the storage size. For example, in order to represent a district with 100 similar houses, a straightforward method is that we first build one mesh to represent one house and then build the other 99 houses by replicating the mesh 99 times with spatial transformations. This method is very simple. However, it consumes a great amount of memory by replicating the mesh multiple times. To solve this problem, most rendering engines adopt a reference, where we first build a mesh for a house, then build 100 nodes. Each node includes a reference to the mesh and the related spatial transformation. In this way, we only need to store one mesh and use references to realize reuse of the meshes multiple times. The references in different engines are realized in different ways, which can be achieved by ID, address or handles.

In order to build the spatial relations of the nodes in a scene graph, we need to build the spatial index to a scene graph. The most often used spatial indices are BSP tree, quad tree, octree and kd tree. With a spatial index we can quickly determine the spatial relations between entity and ray / line, entity and entity, entity and view frustum, including intersection, disjoint or enclosed. With such accelerations, we can obviously improve the time efficiency of scene graph operations, such as object selection and collision detection.

Most rendering engines provide one module, a scene graph manager (different rendering engines may have different names), to help manipulate scene graphs, so as to create, modify, reference, duplicate, rename, search, delete scene graph nodes. The functions of a scene graph manager can be roughly classified into the following categories:

(1) Node lifetime management. This is mainly for creating, duplicating and deleting nodes.

(2) Node field management. This is to provide get / set functions of nodes' fields. Furthermore, it provides more complex field updating functions, such as changing the node's name, which requires solving the name conflicts problems. Applying a transformation such as translation, rotation, scale, or their combinations to nodes is the basic but important function in this category. For some powerful rendering engines, adding and deleting user-defined fields are supported.

(3) Node relation management. This is mainly for grouping / ungrouping nodes, referencing / dereferencing nodes, etc., and collapsing nodes.

(4) Spatial index management. This is to construct and update the spatial index of a scene. Since there are several kinds of spatial index, such as binary partition tree, kd tree and octree, one rendering engine usually implements one spatial index. A local update of the spatial index for dynamic scenes is very crucial for a large scene, for it will save much computational expense.

(5) Query utility. This is for finding nodes by name, type, bounding volume, intersecting ray or other information.

(6) Loader and serializer. Almost every rendering engine has one module to load the scene graph from files. It checks the syntax of the contents of files (some even check the semantics), creates nodes and assembles them to form a scene

graph in the host memory. Most rendering engines tend to define one intrinsic file format, and provide a plug-in of third-part modeling software, such as 3D Studio MAX, Maya, to export their own file. Some of them provide tools to convert other formats into it. Corresponding to the parser module, a rendering engine has a serialize module, which serializes the whole or part of the scene graph into a stream.

1.2 Scene Graph Traverse

The rendering engine works in a cycling manner. For each cycle the rendering engine sets the camera parameters and traverses the scene graph once, during which time it finishes the rendering for one frame. So we only need to set the camera parameters according to the walkthrough path to realize the walkthrough of a scene.

There is one specific module, which we call a traversal manipulator, responsible for scene graph traversal. The traversal manipulator traverses the scene graph node by node. Among the operations done to the nodes, the animation controllers are the most important. They update the states of animated nodes according to the time.

To represent dynamic objects in a virtual world, such as moving vehicles, light flicker, a running athlete and so on, some rendering engines provide various animation controllers, such as a keyframe animation controller, skeletal animation controller, particles controller, etc. Each animated node could have one or several controllers attached. In the traverse of a scene graph, the attached controllers are visited and have the opportunity to execute some codes to make the state of the object up-to-date.

After the operations on the nodes have been done, the traversal manipulator determines which nodes need to be rendered and puts these nodes in a rendering queue. The two most important decisions to made are as follows:

(1) Visibility Culling. By using visibility culling, potentially visible objects are selected and sent to the primitive render, so as to avoid those definitely invisible objects consuming rendering resources. Therefore, visibility culling is an important rendering acceleration technique and is very effective for in-door scenes.

(2) Level-of-detail selection. Level-of-detail technique uses a basic idea to reduce the rendering computation, so that the objects close to the viewpoint are rendered finely and the objects far away from the viewpoint are rendered coarsely. Powered by level-of-detail techniques, one renderable object usually has many versions, each of which has a different level of detail. During the traverse, for each renderable object with LOD, an object version with a proper level of detail is carefully selected according to the distance and viewing direction from the viewpoint to the object, so that the rendering results of the selected object version look almost the same as that of the original object.

1.3 Rendering Queue

Through the traversal manipulator, the to-be-rendered scene graph nodes are stored in the rendering queue and delivered to the render module. The render arranges the nodes in the rendering queue in a proper order, which may be spatial relations from front to back or from back to front, or material types. Most rendering engines regard the underline graphic rendering pipeline as a finite state machine. They arrange the rendering order according to the goal to reduce the switch times of state machines, which improves the rendering speed with the precondition of rendering accuracy.

1.4 Rending Modle

After rearranging the order, the render calls a proper rendering process according to the node types. Generally speaking, there is at least one rendering process for each renderable object, such as triangle meshes, billboards, curves, indexed face sets, NURBS and text. The rendering engine uses one module to manage these rendering processes, which is called a render. The so-called rendering process is actually a set of algorithms, which break down the rendering for a renderable object to a series of rendering statements supported by a bottom graphics library (like OpenGL or Direct3D). A render is not just a simple combination of a set of rendering processes. It has a basic framework to coordinate, arrange and manage the rendering queue and rendering processes. In addition, it includes a series of public rendering utilities, to reduce the difficulty of developing rendering processes for different nodes. The core parts of the render are:

(1) Texture mapping module. This handles a variety of texture mappings, such as multi-texturing, mipmapping, bump mapping, displacement mapping, volumetric texture mapping, procedural texture mapping, etc. Some of the rendering engines also provide texture compression/decompression, texture packing, texture synthesis and other advanced texture related functions.

(2) Shading module. This calculates reflected or refracted light on the object surface covered by a certain material and lit by various light sources, such as point-like omni-light sources, line-like light sources, spotlight sources, directional light sources, surface-like light sources, environmental light sources, etc. The module supports several kinds of illumination models, such as the Phong model, the Blinn model, the Cook and Torrance model, and so on. Adopting a different illumination model usually requires a different appearance model. At the runtime stage, rendering engines only support local illumination, for global illumination is computationally very expensive to achieve in real-time. To simulate global illumination, lightmapping is used by many rendering engines. However, lightmapping is limited to showing diffuse components in static lighting scenarios. Nowadays, precomputed radiosity transfer techniques provide a new way to show

objects with a complex appearance model in dynamic lighting conditions.

(3) Shadows module. As a shadow is a phenomenon of lighting, shadow computation rigorously should be a feature of a lighting and shading module. Nevertheless, if we consider the light source as a viewpoint, the shaded places can be considered as the places invisible to the light source. Therefore, shadow computation by nature is a visibility determination problem, which is a global problem depending on the spatial relations of the entire scene, including object-object and light-object relations. Due to the complexity of this problem, it becomes a separate module in most rendering engines.

Besides the above important modules, some rendering engines provide a series of assistance modules, such as:

(1) Observer controller: To control the observer's position, direction, field of view, motion speed/angular speed, motion path and the characteristics of image sensors.

(2) Special visual effects: To simulate the effects of ground/water explosion, explosion fragments, flashes from weapon firing, traces of flying missiles, rotation of airscrew, airflows of rotating wings, smoke and flames, etc.

(3) Display environment configuration: To support the display devices of CAVE, Powerwall etc. Users can configure the number of displays, their arrangement styles and the stereo eyes' distance. This also supports non-linear distortion correction, cylinder and planar projection display and the edge blending of multi-displays.

此为试读,需要完整PDF请访问: www.ertongbook.com