[quote]

[align=center][b][color=orangered][size=200%]' 免 责 声 明 '[/size][/color][/b][/align]

======

&#8195;&#8195;[b][size=150%]该[/size]教材的出版时间是2009年，由于游戏与代码的迭代特性，本书的内容可能已经与现在你实际接触到的游戏或代码内容有所差异，有些内容可能已经过时，甚至是被淘汰“过期的”，请大家自己斟酌参考。若出现由于原文与本帖翻译有所不同而导致的错误或损失，概不负责。[/b]

[/quote]

[quote]

[align=center][b][color=orangered][size=200%]' 扇 贝 碎 碎 念 '[/size][/color][/b][/align]

======

[b][size=150%]这[/size][/b]是国外教材《Beginning Lua with World of Warcraft Addons》的最后一个章节。我已经出了我到目前为止所翻译的几个章节的合集，大家可以自行浏览[tid=24502396][color=orangered][合集]Lua与魔兽世界插件——《Beginning Lua with World of Warcraft Addons》[/color][/tid]

大家好啊，很惭愧消失了半年。原因嘛，主要是遇到了让我很受挫的事情，心态上有点没调整过来，一时间不知道该怎么办，也就逃避了大半年。这大半年里就先投入生活，先让自己睡得安稳，吃得开心。

[del]印象里，这章节我似乎已经发过了？也可能是我记忆发生了错乱。[/del][b]这章节其实本身已经跟《魔兽世界》的插件没有太大的关系了，讲得基本是Lua在其它游戏或游戏外的应用。[/b]本身其实可以不翻译，只是为了保持风格一致，让整个翻译工作有头有尾，有兴趣的自己看看就行。

说下翻译工作，其实有好几个章节我在半年前就已经翻译完成了，例如第三章、第十四章，还有最近翻译的第十三章，只是还没有发出来。当时我去出差回来，诱发了不可抗力，被隔离14天。本想着过段时间再发，但之后却让我陷入了长达半年的自闭。其实从翻译到校对到排版，这个过程是稍稍有些枯燥的。完全没有写插件最后在游戏中呈现效果那样令人热血澎湃。而且最大的问题，我翻译完之后有些地方会重新看一遍，如果发帖了之后，我不但需要更改译文，还需要更改发帖的内容，这也无形之中增加了工作量。所以我会把译文放一段时间，简单看一遍再发帖。

[size=110%][b]如果有人在阅读过程中，如果发现哪里翻译错误，或错别字需要更正，请发评论或发私信告知，谢谢大家。[/b][/size]

[/quote]

[quote]

[align=center][b][color=orangered][size=200%]' 目&#8195;&#8195;录 '[/size][/color][/b][/align]

======

[size=180%][b]&#9632; 第十四章 Lua的其它用途[/b][/size]

[size=130%][b]&#9679; 使用Lua定制SciTE[/b][/size]

&#9675; [b]“Hello, World”和SciTE[/b]

&#9675; [b]事件处理程序[/b]

&#9675; [b]SciTE文档[/b]

[size=130%][b]&#9679; 《战锤Online》中的Lua[/b][/size]

&#9675; [b]《战锤Online》中的“Hello, World”[/b]

· .mod文件

· Lua文件

· 事件处理程序

&#9675; [b]文档[/b]

[size=130%][b]&#9679; Lua和Lugre[/b][/size]

&#9675; [b]配置Lugre[/b]

&#9675; [b]创建一个简单的3D应用程序[/b]

· 创建树

· 天空

&#9675; [b]文档[/b]

[size=130%][b]&#9679;总结[/b][/size]

[/quote]

======

[align=center][color=orangered][b][size=200%]&#9632; 第十四章 Lua的其它用途[/size][/b]

[size=130%][b](Other Uses for Lua)[/b][/size][/color][/align]

[b][size=150%]现[/size][/b]在你已经掌握了编写强大的《魔兽世界》插件所需的绝大部分东西，但你学到的不仅仅是如何编写《魔兽世界》插件。你学习了编程语言Lua，除了《魔兽世界》之外，还有更多的应用程序和游戏在使用它。一个使用Lua的应用程序的不完整列表可以在Wikipedia上找到[url]http://en.wikipedia.org/wiki/Lua\_(programming\_language)#Applications[/url]。

有许多游戏，例如《战锤Online》、《孤岛危机》、《孤岛惊魂》和《S.T.A.L.K.E.R》都使用了类似于《魔兽世界》的Lua。除了《魔兽世界》，使用了Lua界面的最有趣的游戏是《战锤Online》，因为它也支持与《魔兽世界》非常相似的插件。我将在这里向你展示一个简单的《战锤Online》的“Hello，World”插件。

另一个特别有趣的应用程序是SciTE，我在本书开头介绍过这个编辑器。你可以根据你的需要使用Lua来定制它，我将在本章中展示怎么去做。我们将编写一个在编辑器中更改Lua代码缩进的功能，当你从WoWWiki的文章或论坛帖子等地方复制和粘贴Lua代码时，这个功能会很有用。

在这一章里，我要给你们展示另一个项目。Lugre是一个绑定了Lua的OGRE 3D图形引擎，它是一个功能齐全的3D图形引擎，可以用来编写你自己的游戏。Lugre允许你从Lua使用它的所有特性。我将在此向你展示一个简单的例子，因为我认为这是使用Lua最酷的项目之一。

======

[color=blue][b][size=160%]&#9679; 使用Lua定制SciTE [/size]

[size=130%](Customizing SciTE with Lua)[/size][/b][/color]

你可能已经在使用这个编辑器作为插件的IDE了。如果你一直在使用其他IDE或编辑器，可以在代码下载中找到我的定制版本。该编辑器基于Scintilla，这是一个被许多编辑器使用的开源编辑器引擎，最著名的是SciTE、Notepad++和Code::Blocks。SciTE是高度可定制的，你可以通过修改配置文件来更改几乎所有内容，而且几乎整个Scintilla API都是向提供Lua脚本的用户公开。

让我们先看一下配置文件。你可以通过在选项(Options)菜单中单击“打开全局选项文件(Open Global Options File)”来打开该文件。我已经在这里添加了两个用户定义的命令：一个简单的“Hello,World”脚本和一个插入当前时间的脚本。你可以在配置文件的末尾找到这两个命令的定义。

[quote]

## Hello, World

command.name.1.\*Hello World from SciTE

command.1.\*HelloWorld

command.subsystem.1.\*=3

command.mode.1.\*=savebefore:no

## Insert Timestamp

command.name.2.\*=Timestamp

command.2.\*=Timestamp

command.subsystem.1.\*=3

command.mode.1.\*=savebefore:no

command.shortcut.2.\*=Ctrl+1

[/quote]

这里我不打算详细介绍SciTE配置命令，但是你可以在[url]http://www.scintilla.org/SciTEDoc.html[/url]上找到完整的文档。这段代码主要创建了两个显示在Tools菜单中的命令。第二个选项使用Ctrl-1作为快捷键，而第一个选项没有定义快捷键。它将自动使用Ctrl-1，因为这是第一个自定义命令。这两个命令之后调用全局Lua函数HelloWorld和Timestamp，它们是在启动脚本startup. Lua中创建的。让我们从一个简单的“Hello，World”脚本开始。

[b][size=130%]&#9675; “Hello, World”和SciTE(Hello, World with SciTE)[/size][/b]

文件startup.lua位于SciTE的安装目录中，但是你也可以通过单击开打Lua脚本(Open Lua Startup Script)下的选项(Options)来打开它。我们可以在这里创建函数HelloWorld：

[code=lua]

function HelloWorld()

print(“Hello, World”)

end

[/code]

你现在可以按Ctrl-1(或从工具菜单中选择Hello, World)来执行它，不需要重新启动SciTE。当你保存文件时，它会自动检测你何时修改启动脚本(startup script)，并在瞬间重新加载。我们得到的输出只是在SciTE的默认输出区域中的“Hello, World”

但是打印到标准输出是很简单的，但是在当前打开的文档中插入文本会更有趣。SciTE在这个启动脚本的全局命名空间中提供了一些对象。其中一个对象是editor。它提供了许多方法，可用于插入文本、读取文本或修改现有文本。例如，我们可以使用方法editor:AddText(text)在当前光标位置插入一些内容。

[code=lua]

function HelloWorld()

editor:AddText(“Hello, World”)

end

[/code]

现在，当我们按下Ctrl-1或从菜单中选择命令时，SciTE将“Hello，World”插入到我们的文件中。当然，这里有所有常用的Lua函数，所以很容易编写将当前日期和时间插入文档的命令。

[code=lua]

function Timestamp()

editor:AddText(os.date())

end

[/code]

但你并不需要点击菜单中的命令来执行Lua函数，你还可以定义事件处理程序，这些处理程序在文件保存等事件中被调用。

[b][size=130%]&#9675; 事件处理程序(Event Handlers)[/size][/b]

通过在事件后命名全局函数，可以创建充当事件处理程序的函数。例如，当你双击编辑器或输出面板时，OnDoubleClick事件发生。将以下函数添加到启动脚本中，以便在用户每次双击时显示一条信息：

[code=lua]

function OnDoubleClick()

print(“OnDoubleClick”)

end

[/code]

每次双击，它都会打印OnDoubleClick，这是一个真实有效的演示，但不是一个真正有用的应用程序。其他事件处理程序可能更有价值，比如OnBeforeSave，它在保存文件之前执行，并接收被保存的文件的名称作为参数。在将文件写入磁盘之前，可以使用它进行一些检查或修复。

例如，我们可以在保存文件之前插入“最后修改(last modified)”的时间戳。下面的OnBerforeSave处理程序通过使用遍历编辑器的所有行来实现这一点。LineCount用来确定行数而editor:GetLine(i)用来检索行。它查找模式$Modified.-$，并使用editor:SetSel(selStart, selEnd)和editor:ReplaceSel(newText)方法将其替换为$Modified.timestamp$。

[quote]

注意：这个函数中的字符串不能使用$Modified…$，否则事件处理程序将在编辑文件时修改自己，这将非常的烦人。因此，下面的代码使“\36($的ASCII码)”代替“$”来防止这种情况。

[/quote]

事件处理程序使用属性editor.CurrentPos在进行选择之前保存插入符号的位置，并使用方法editor:GotoPos(pos)在更新文件后恢复插入符号的位置。这可以防止脚本改变插入符号的位置并滚动到$Modified$。

[code=lua]

function OnBeforeSave(file)

local oldPos = editor.CurrentPos -- save the old position to prevent scrolling

-- lines are zero-based in SciTE

for i = 0, editor.LineCount - 1 do

local line = editor:GetLine(i)

if line then -- line is sometimes nil in large files

local startPos, endPos = line:find(“\36Modified.-%\36”)

if startPos and endPos then -- text in current line?

-- get the absolute position of the line in the document

local lineStart = editor:PositionFromLine(i)

-- select text and replace it, the editor position is 0-based

-- but the result frome string.find is 1-based

editor:SetSel(lineStart + startPos - 1, lineStart + endPos)

editor:ReplaceSel(string.format(“\36Modified:%s\36”, os.date()))

end

end

end

editor:GotoPos(oldPos) -- restore old position

end

[/code]

这个简单的事件处理程序允许你在文件中的任何地方放置&Modified&，每当你保存文件，它将自动添加并更新一个时间戳。

[b][size=130%]&#9675; SciTE文档(SciTE Documentation)[/size][/b]

在SciTE上有很多很好的可用文档：

[url]http://scintilla.sourceforge.net/SciTELua.html[/url]：官方文档，它只涵盖了基础知识。

[url]http://lua-user.org/wiki/UsingLuaWithScite[/url]：一个非常好的教材和文档。

[url]http://lua-users.org/wiki/SciteScripts[/url]：很多有用的Lua脚本可用在SciTE中使用。

[url]http://scite-interest.googlegroups.com/web/ScintillaSciteDoc.html[/url]：所有可用对象及其方法/属性的完整文档。

我们的下一个主题是另一个MMORPG(大型多人在线角色扮演游戏)，《战锤Online》(WAR)。它的界面是用Lua编写的，可用通过插件进行扩展。

======

[color=blue][b][size=160%]&#9679; 战锤Online》中的Lua [/size]

[size=130%](Lua in Warhammer Onlin)[/size][/b][/color]

在本节中，你将看到一个简单的“Hello，World”插件是如何在《战锤Online》([url]http://www.warnammeronline.com[/url])中工作的。《魔兽世界》和《战锤》的用户界面有许多相似之处，《魔兽世界》插件的程序员很容易就能在战锤中开始开发。在我看来，《战锤》界面API就像是《魔兽世界》API的粗糙版本，许多功能与《魔兽世界》类似，但使用起来更困难或更复杂。

[b][size=130%]&#9675; 《战锤Online》中的“Hello, World”(Hello, World in Warhammer Online)[/size][/b]

插件被放置在Interface\AddOns\&#60;addon name&#62;，你必须为你的第一个插件手动创建这个文件夹。为我们的“Hello, World”插件创建一个名为HelloWorld的文件夹。如果我们现在正在为《魔兽世界》编程，那么下一步将是创建一个.toc文件。《战锤Online》使用的是.mod文件。

[b]· .mod文件[/b]

《魔兽世界》的.toc文件和《战锤》之间的主要区别是，.mod文件是用XML编写的。在HelloWorld文件夹中创建一个名为HelloWorld.mod的文件，并在其中放入以下XML：

[code=xml]

<?xml version=”1.0” encoding=”UTF-8”?>

<ModuleFile xmlns:xsi=”http”//www.w3.org/2001/XMLSchema-instance”>

<UiMod name=”HelloWorld” version=”1.2.1” autoenabled=”true”>

<Description text=”A Hello, World AddOn”/>

<Author name=”You!”/>

<Files>

<File name=”HelloWorld.lua”/>

</Files>

</UiMod>

</ModuleFile>

[/code]

XML应该是不言自明的。创建插件的游戏版本存储在UiMod元素的version属性中。这个属性相当于《魔兽世界》的interface属性。如果这个版本号低于游戏的当前版本(目前是1.2.1)，游戏会抱怨插件过时了。

XML还包含例如插件的描述和作者这样元数据。之后它会加载文件HelloWorld.lua，我们接下来将编写该文件。

[quote]

小贴士：当你在战锤中更改.mod文件时，没有必要重新开始游戏。只需要输入/reload就可用重新加载UI。

[/quote]

[b]· Lua文件[/b]

创建一个“Hello, World”插件最简单的方法就是在这个文件中编写一个显示“Hello，World”的代码。你可能想在哪里写入print(“Hello,World”)，但print不可用。我们必须使用EA\_ChatWindow.Print(msg)方法来打印消息。

但是这个函数不接收普通的Lua字符串，我们必须把它转换成UCS-2编码的字符串。这是战锤中使用的字符编码，它意味着每个字符由两个字节组成。有一个简单的函数可用接收一个普通字符串，并将其转化成UCS-2，以便显示：L。回想一下，对于只接收单个字符串或表的函数调用，不需要包含圆括号。因此我们可用简单地将以下代码写入Lua文件并使用/reload重新加载界面：

[code=lua]

EA\_ChatWindow.Print(L“Hello,World!”)

[/code]

在重新加载UI后，我们现在可以在聊天框中看到“Hello, World”了。

[quote]

小贴士：在打开的菜单中使用命令/debug并启用日志记录，可以查看战锤中的Lua错误。

[/quote]

现在，你可能期望“Hello, World”插件的下一步是添加一个斜杠命令。但《战锤Online》中的斜杠命令非常复杂。添加斜杠命令的唯一方式是钩住(hooking)游戏提供的函数。有一个库可以执行这个钩子并提供一个函数来注册斜杠命令：LibSlash，它可以在war.curse.com上下载。它很容易使用，并且有自己的文档，所以我在这里跳过斜杠命令。事件处理程序会更有趣。

[b]· 事件处理程序[/b]

在《战锤Online》中有两种事件处理程序，在插件.mod文件中定义的事件和游戏事件。前一种事件类型相当于《魔兽世界》中的脚本处理程序，而后者类似于《魔兽世界》游戏的事件处理程序。

让我们从.mod文件中的事件开始。只有三个事件处理程序可用。

OnInitialize：当加载插件时调用。

OnShutdown：在离开游戏或重新加载界面前调用。

OnUpdate：调用每一个框体(frame)

为了测试OnInitialize处理程序，需要将以下元素添加到.mod文件的UiMod元素中。

[code=xml]

<OnInitialize>

<CallFunction name=”HelloWorld\_Initialize”/>

</OnInitialize>

[/code]

这只是调用存储在全局变量HelloWorld\_Initialize中的函数。让我们在Lua文件中创建这个函数。

[code=lua]

function HelloWorld\_Initialize()

EA\_ChatWindow.Print(L”Hello, World from OnInitialize!”)

end

[/code]

这将在重新加载界面时在聊天框中生成一条消息。

第二种类型的事件处理程序可以用RegisterEventHandler(event, func)函数注册。event参数不是字符串，而是标识事件的数字。所有事件及其对应的id都存储在表SystemData.Events中。同样的，func不是函数，而是一个字符串，它保存了包含函数的全局变量的名称，该全局变量在事件发生时被调用。也可以传递一个“someTable.key”格式的字符串来调用全局变量someTable表中key下存储的函数。

让我们使用事件CHAT\_TEXT\_ARRIVED进行测试，它会在每次收到聊天消息时触发。完整的活动列表可以在[url]http://www.thewarwiki.com/wiki/Event\_List[/url]上找到。我们不能将事件名作为字符串传递，因此需要从表SystemData.Events中获取事件的ID，之后我们的函数调用看起来就像这样：

[code=lua]

RegisterEventHandler(SystemData.Events.CHAT\_TEXT\_ARRIVED, “HelloWorld\_OnChat”)

[/code]

现在，我们可以创建函数HelloWorld\_OnChat，每次事件发生时都会调用该函数。可能有人希望这个函数接收包含聊天消息的发送者和文本参数，但事实并非如此。事件参数存储在GameData.ChatData中。name参数存储发送消息的玩家(或NPC)，text存储实际的消息。让我们编写一个函数，简单地在你的聊天中框显示消息。

这听起来很简单，但是这个表中的名称和文本字段不是字符串，它们是wstring。wstring是战锤引入的数据类型，它是多字节UCS-2编码的字符串之一，我们也需要用函数Print。问题是你不能在关联(concatenations)中混合普通字符串和wstring。一个可能的解决方案似乎是使用string.format，但这对wstring根本不起作用。

一个正常工作的显示消息的函数是这样的：

[code=lua]

function HelloWorld\_OnChat()

local name = GameData.ChatData.name

local msg = GameData.ChatData.text

EA\_ChatWindow.Print(L”<”..name..L”>”..msg)

end

[/code]

游戏不会在自动在wstring和普通字符串之间转换，你必须一直手动去做这件事。你可以使用wStringToString将wstring转换为普通字符串，但如果字符串包含特殊字符，则可能丢失信息。

这些就是在《战锤Online》插件中使用Lua的基本原理。它比《魔兽世界》API稍微复杂一些，特别是字符串又两种不同的类型会非常令人困惑和烦恼。但它仍然是Lua，因此理解和学习API并不难。

[b][size=130%]&#9675; 文档(Documentation)[/size][/b]

如果你想了解更多关于《战锤Online》的插件信息，有很多非常好的网站：

[url]http://www.thewarwiki.com[/url]：这个wiki相当于《战锤Online》中的WoWWiki。

[url]http://war.curse.com/[/url]：Curse不仅适用于《魔兽世界》插件，你也可以在那里找到许多《战锤Online》插件。

现在你知道了如何在两款不同的游戏中使用Lua，但如果让你自己编写游戏不是更酷吗？你可以通过使用Lua引擎Lugre来实现这一点。

======

[color=blue][b][size=160%]&#9679; Lua和Lugre [/size]

[size=130%](Lua and Lugre)[/size][/b][/color]

Lugre([url]http://lugre.schattenkind.net[/url])为OGRE 3D提供了Lua API，可以使用Lua编写整个游戏。这一节稍微高级一些，需要3D图形的基础知识。我不会在这里解释每一个术语，因为整个主题非常复杂，可以填满整本书。事实上，有一本关于OGRE 3D的好书：由Gregory Junker编写的《Pro OGRE 3D Programming》(Apress 2006)。它没有涵盖Lugre，但你需要了解OGRE 3D如何工作才能使用Lugre，它只是将你的大多数函数和方法的调用转发给相应的OGRE函数和方法。

[b][size=130%]&#9675; 配置Lugre(Setting Up Lugre)[/size][/b]

安装Lugre最简单的方法是从SVN版本库：[url]svn://zwischenwelt.org/lugre/trunk/example[/url]中获取预编译的二进制版本和一个小的示例项目。你可能不知道Subversion (SVN)是做什么的，它是一个版本控制系统，管理项目的源代码。它允许你在一个项目的不同版本之间切换，并且在同步文件的同时让多个程序员在同一个项目上工作。

这意味着你需要一个使用Subversion命令checkout(拉取)的客户端来获取存储在此版本库中的所有代码的所有最新版本。一个很好的Windows客户端是TortoiseSVN，可以在[url]http://tortoisesvn.tigris.org/[/url]上获得。在资源管理器的快捷菜单中会有一些新命令，其中之一就是SVN Checkout。为Lugre创建一个新文件夹并运行此命令。当它要求你输入存储器(repositor)输入[url]svn://zwischenwelt.org/lugre/trunk/example[/url]。

[quote]

注意：Subversion不仅对例如Lugre这样相对较大的项目有用。我对每一个插件都使用它，比“Hello, World”这样的例子，因为它是一个难以置信的有用的工具。Curse.com为插件提供免费的Subversion版本库。也有关于设置它的教程，因此使用TortoiseSVN是相当容易的。

[/quote]

Lugre附带了一个小的示例项目，你可以运行文件夹bin中的example.exe文件来启动它。它显示了一棵树，没有什么令人印象深刻的。但是考虑到负责创建和管理这个场景的整个代码都是用Lua编写的。下一个示例展示了这是如何工作的。

[b][size=130%]&#9675; 创建一个简单的3D应用程序(Creating a Simple 3D Application)[/size][/b]

负责这个应用程序的代码可以在main.lua文件中找到。我们不会删除整个文件，因为它还包含了许多枯燥的初始化代码，我们将在示例中重用这些代码。

打开文件并导航到Main函数，这是Lugre在程序启动时调用的主函数。在函数的开头找到以下两行：

[code=lua]

----- your init code here -----

Bind(“v”, function(state) Client\_TakeScreenshot(gMainWorkingDir..”screenshots/”) end

[/code]

现在删除这些行之间的所有东西(保持调用Bind，以便你可以通过按V进行截图)和下面的行：

[code=lua]

-- mainloop

while (Client\_IsAlive()) do MainStep() end

[/code]

重要的是不要删除这两行，因为这是程序的主循环。它反复调用MainStep函数，直到你关闭程序。MainStep可以在Main函数下面找到。这个函数调用其他函数，这些函数处理用户输入和绘制框体(frame)。这个函数不会改变任何东西，我们将只替换先前删除的代码。

现在我们将创建一个简单的场景，类似于刚开始的例子，显示一些不同参数的树和一个简单的场景(skybox)

[b]· 创建树(Creating Trees)[/b]

从CaduneTreeParameters对象创建树，该对象存储树的各种属性。多个树可以使用相同的CaduneTreeParameters对象来创建多个外观相似的树。这些对象不是OGRE 3D的直接部分，它们是由OGRE插件Cadune Tree提供的。但是Lugre已经为OGRE提供了很多扩展和插件，让你的工作更轻松。

树本身包含两个不同的GFX对象，一个是茎，一个是叶。这两个物体需要放置在我们场景中的特定位置。我们将把树的创建放在一个小的辅助函数中，这样我们就可以快速创建不同的树，而不需要复制和粘贴大量的代码。在main函数的上方创建以下函数，因为它需要在那里可见：

[code=lua]

local function CreateTree(leaves, leafScale, leafMaterial, x, y, z)

local p = CreateCaduneTreeParameters()

p:SetNumLeaves(leaves)

p:SetLeafScale(leafScale)

p:SetLeafMaterial(leafMaterial)

local s = CreateCaduneTreeStem(p)

s:Grow()

local gfx\_stem = s:CreateGeometry()

local gfx\_leac = s:CreateLeaves()

gfx\_stem:SetPosition(x, y, z)

gfx\_leav:SetPosition(x, y, z)

return s

end

[/code]

该函数接受三个参数，它们决定树叶的外观，之后是三个参数，它们定义小场景中树的位置。现在，我们可以通过将下面一行放在main函数中先前删除旧示例代码的地方来创建第一个树。

[code=lua]

local s1 = CreateTree(10, 3, “Leaves/Ivylite”, -10, -10, 10)

[/code]

这在我们场景的左侧区域创建了一颗漂亮的树。你可以使用这些参数来修改树的外观。添加以下一行，在屏幕的右半部分创建第二课树：

[code=lua]

local s2 = CreateTree(3, 5, “Leaves/Ivylite”, 10, -10, 10)

[/code]

这将创建一个具有不同类型的叶子和不同参数的树。我们需要一个带有太阳和月亮的天空作为我们的场景。

[b]· 天空(The Sky)[/b]

我们在这里使用的对象和树对象不一样，并不是OGRE 3D的一部分，而是Lugre中包含的附加库的一部分，在这里是Caelum。它不只是简单地展示了一个完整的宇宙。它会根据时间显示太阳和一些云或月亮和星星。时间是可以设定的，并且可以使用指定的倍增器自动前进，这样我们就可以很容易地实现昼夜效果，而不需要编写大量的代码。

将下面几行放在创建这两个树的代码下面：

[code=lua]

local caelum = CreateCaelumCaelumSystem(

CAELUM\_COMPONENT\_SKY\_DOME +

CAELUM\_COMPONENT\_SUM +

CAELUM\_COMPONENT\_CLOUDS +

CAELUM\_COMPONENT\_MOON +

CAELUM\_COMPONENT\_IMAGE\_STARFIELD

)

[/code]

现在，在我们的场景中有一个简单的天空，但我们的树木拾取了它们的颜色，它们现在是灰色的。我们需要设置一些参数来定义Caelum对象如何在场景中控制环境光和雾。添加以下代码来为我们的树获得真实的颜色。

[code=lua]

caelum:SetManageSceneFog(true)

caelum:SetSceneFogDensityMultiplier(0.0001)

caelum:SetManageAmbientLight(true)

[/code]

场景看起来仍然是静态的，因为我们还没有定义时间尺度。这意味着时间是实时运行的，所以你需要等待几个小时，直到我们的小宇宙变成夜晚。但我们可以通过调用连接在Caelum上的UniversalClock对象的SetTimeScale来加快速度。

[code=lua]

caelum:GetUniversalClock():SetTimeScale(1000)

[/code]

你可以使用这些参数来达到不同的效果。我们已经创建了一个简单的场景，但你现在可以预见使用这个引擎可以创建整个游戏。

[b][size=130%]&#9675; 文档(Documentation)[/size][/b]

如果你想了解更多关于Lugre和OGRE的知识，有很多很好的网站：

[url]http://lugre.schattenkind.net/index.php/Main\_Page[/url]：官方的Lugre wiki，包含了一个教程，介绍了如何用Lugre创建一个简单的乒乓球游戏(Pong game)。

[url]http://www.ogre3d.org[/url]：OGRE 3D项目的网站，Lugre使用的3D引擎。有许多令人兴奋的教程，以及API和插件的文档。

======

[color=blue][b][size=160%]&#9679; 总结 [/size]

[size=130%](Summary)[/size][/b][/color]

[b][size=150%]在[/size][/b]这一章中，你可以看到Lua不仅仅适用于《魔兽世界》。它作为一种脚本语言被嵌入到许多游戏和应用程序中。所有平台的语言都是一样的，只有提供由函数、变量和对象组成的API不同。

但是Lua不仅可以在另一个应用程序中使用。有一些框架允许你用Lua编写完整的独立应用程序。在这里我们看到了Lugre框架，它允许你只用Lua编写完整的游戏。但是，还有成百上千的其他框架为Lua程序添加了令人兴奋的功能。如果编写《魔兽世界》插件变得很无聊，那么这里使用Lua的方法有很多。你可以在[url]http://lua-users.org/wiki/LibrarieAndBindings[/url]上找到包含大量库和绑定(binding)的列表，这些库和绑定允许你使用其他库或框架。