# 角色动画简介

您好，亲爱的读者，欢迎阅读本书关于角色动画！ 我希望你喜欢它并发现它很有用。 在本章中，我将通过查看一般的角色动画以及相同主题的简要历史来让您慢慢开始。 您将不会在本章中进行任何编码，但最后我将简要概述本书中使用的编码约定。 在第一章中，您将找到以下内容：

乍一看，这个有点愚蠢的问题看起来很容易回答，但是真的吗？ 维基百科将其定义如下：

“在涉及动画作品中一个或多个角色的动画中，角色动画是动画过程中的一个专门领域。”

-----维基百科

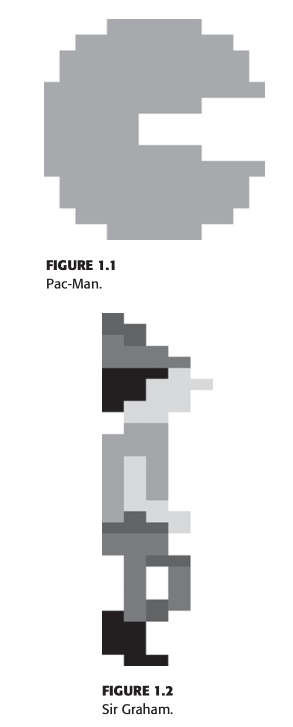
动画制作，我想游戏制作也属于这一类。然而，我可能会试图在“让角色以真实的方式移动“的方向上定义它。虽然，我认为这更好地回答了这个问题-“什么是角色动画的目标”。

历史上，角色的绘制(或像素化)和动画是通过制作多张图片来显示角色在一个稍微不同的姿势。这些图片会循环播放，给人一种运动的感觉。使用今天的显卡，可以拥有完整的三维角色，并使用本书中介绍的各种技术为它们制作动画。

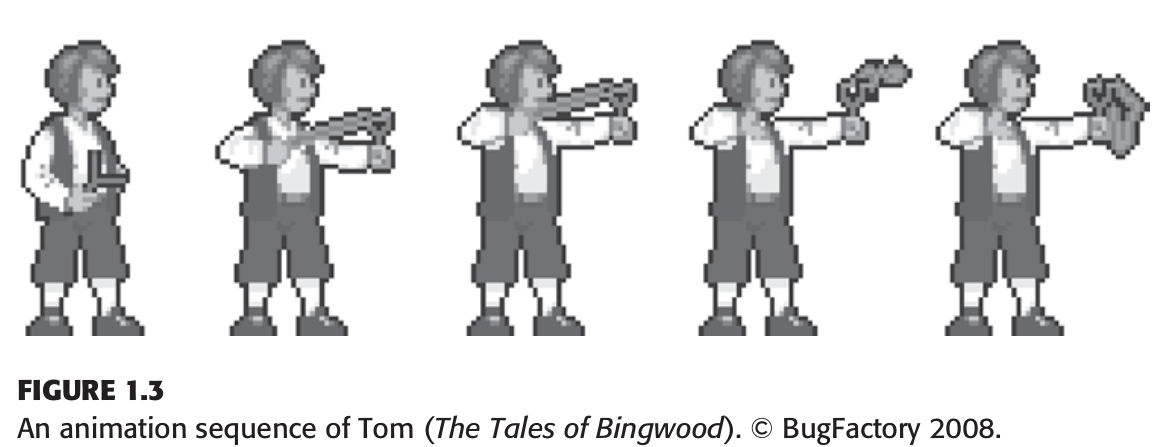
每年都有新的动画角色模型出现，推动了该领域的发展。本书所涉及的技术绝不是最前沿的。相反，它们是所有(或大多数)前沿技术的基础技术。骨骼动画、变形、布偶物理仿真、逆运动学等技术已经存在了很长时间(至少在游戏革新方面)。

不过，在这本书的最后，你将拥有所有的工具，你可以创建属于的自己的人物动画逼真的游戏。让我们从头开始吧!向我们这个时代第一个著名的计算机游戏角色之一打招呼:Pac-Man(图1.1)。

这个28 \* 28像素的角色(由Namco开发)于1980年在日本发行，至今仍是最著名的电脑游戏。这个角色(看起来更像少了一片披萨)慢慢地让位于更多的类人角色。四年后，Sierra online发布了Kings Quest: Quest for the Crown，来看看Graham爵士(图1.2)。



格雷厄姆爵士的像素可能不会比《吃豆人》多多少，但至少他的色彩更丰富，拥有一套很棒的动画。直到80年代末，字符都沿着同样的路线发展，像素和/或颜色都在稳步增加。1987年，LucasArts开发了SCUMM引擎的第一个版本(Maniac Mansion的脚本创建工具)，并随之发行了几款冒险游戏，包括Maniac Mansion、Monkey Island、Loom等等。这个时代(80年代末到90年代初)的角色几乎具有相同的复杂性——图1.3显示了一个例子。

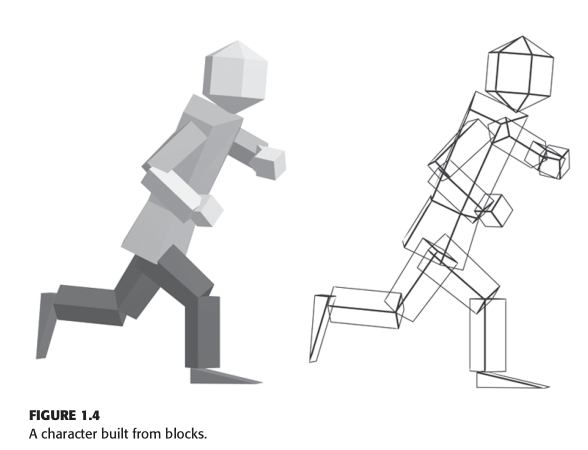


90年代，随着新十年的到来，第一批3D游戏带来了一系列全新的问题。一些更著名的游戏包括《沃尔芬斯坦3D版》(Wolfenstein 3Dand)，后来的《末日》(doomand)——尽管这些游戏并不能真正被称为真正的3D游戏，因为它们仍然使用2D精灵去创造敌人和角色(通常从8个角度绘制，这取决于它们面向玩家的方向)。第一个真正的3D角色出现在1992年发行的游戏《Alone in The Dark》中。它以全3D的角色为特色，带有插值动画。

这些字符的多边形数极低，由几个块(每个肢体一个)组成。图1.4中可以看到这个时代的一个示例字符。

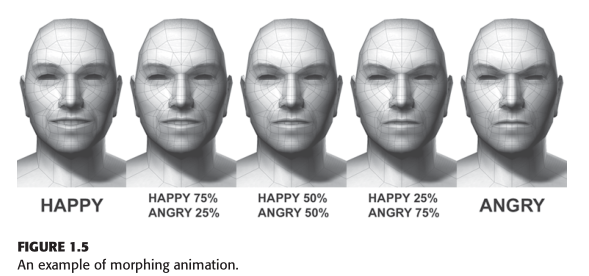
你可以很容易地在这个角色的关节间看到明显的间隙，但在那个时候通常没有模型的光照，而且分辨率非常小，以至于这些间隙通常对玩家隐藏。

几年前，我们达到了1996年，当3dfx推出第一款Voodoo芯片组时，它为大众带来了价格合理的3D加速卡。着名游戏之一是Quake游戏是首批利用这项新技术的。Quake使用顶点变形来制作无缝字符（尽管是低多边形）。



变形动画（或每顶点动画）通过在每个顶点的基础上将两个（或更多个）网格混合在一起来工作。这两个网格需要有相同数量的顶点，并且它们的多边形需要以相同的方式排列，以便这种技术能够工作。每个代表角色姿态的网格都被称为变形目标。每个代表角色姿态的网格都被称为变形目标。可以使用多个变形目标混合最终网格。现在变形动画的主要用途是面部动画。但在过去，它也被用来创建全身角色动画。例如，Quake I和II引擎使用流行但稍微过时的MD2文件格式[Schoenblum07, Leimbach02]中其字符使用这种方法。

有关变形动画的示例，请参见图1.5。在这个图中，只有HAPPY和ANGRY的帧才是实际的目标网格。中间的网格是通过随时间平滑地插入顶点位置来创建的。



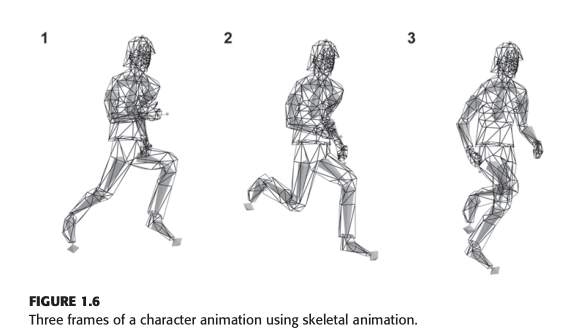
快速转发再次将我们带到了1998年，当时Sierra Studios发布了Half-Life（半条命~）游戏（由Valve开发）。Half-Life建立在Quake引擎的高度修改版本之上。最值得注意的是，游戏开发者添加了一个新的骨骼动画系统，允许他们在不同的角色上重复使用动画。

顾名思义，骨骼动画与骨架的运作密切相关。一般人体约有206块骨头。 所有这些骨骼的状态和位置定义了一个人的姿势。当骨骼从空间的一个位置移动到另一个位置时，周围的肌肉、组织和外层皮肤也随之移动。这个基本思想是骨骼动画的关键。 唯一的区别是，对于电脑游戏，你只对皮肤层感兴趣（即玩家看到的东西）。 在第3章中，您将学习如何“塑造一个角色的皮肤”。

有关蒙皮角色的线框渲染示例，请参见图1.6。请注意皮肤（网格）在移动时如何跟随骨骼。

从Half-Life游戏的发布的日子开始，游戏角色已经获得了更多的多边形，更大的纹理，法线贴图，高级着色器等等，以使它们每年看起来越来越好。但是，基本的基础技术并没有太大变化。

这两种技术 - 骨骼动画和变形动画 - 在游戏开发中被广泛使用，本书将涵盖两者。他们各有优劣。 在本书的最后，您将了解如何创建使用这两种技术的角色 - 例如，用于整体运动的骨骼动画，以及用于面部表情等更微妙的事物的变形动画。



我将本书中使用的示例角色称为“士兵”。看起来又是另一个未来主义英雄形象的力量装甲......好吧，实际上就是这样：另一个未来主义 - 英雄 - 形象 - 动力装甲。

士兵的设计基于老罗马士兵，你可以从肩垫和头盔中发现。

士兵的设计和纹理是由Markus Tuppurainen为我们的冒险游戏《愤怒日》所做的。虽然那款游戏从未完成(是的，是的，我也没能完成我开始的所有游戏)，但这款模型对于这本书仍然有它的用途。重要的是，他拥有所有必要的肢体，一些动画，皮肤网状物（身体和脸部），以及一些静态网状物（头盔和脉冲步枪）。

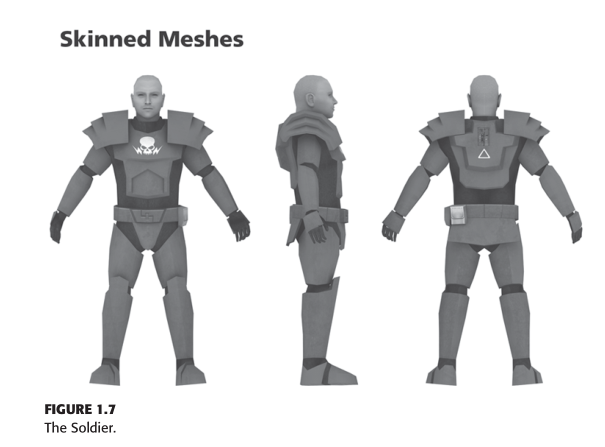
根据今天的标准，模型的复杂程度介于中低范围内：

主体网格:2100多边形

头部网格:1000个多边形

骨骼数量:26

高度:1.8个单位



在本书中，我将使用匈牙利符号标准的一部分（我会尽量保持一致）。本书中的高级着色语言（HLSL）效果将略微偏离此标准，并使用恩格尔着色器书中使用的符号。有关本书C ++示例中使用的编码约定，请参见表1.1。

因此，一般的c++类应该是这样的:

class SomeClass

{

public:

SomeClass();

~SomeClass();

void SomeFunction1(int someParameter);

bool SomeFunction2();

private:

int m\_memberVariable;

float\* m\_pMemberPointer;

};

希望在阅读完本章之后，您已经对角色动画的主题以及我们面前的这一领域的工作有所了解。然而，在短暂的热身之后，是时候开始动手了。在下一章中，您将要看到Direct3D的相关介绍以及创建3D应用程序的必要步骤，其中大部分可能会重复使用。在下一章结束时，您将在屏幕上呈现一个角色（虽然非常僵硬）。然后，在第二章之后，更多的功能将被添加到我们现在有点死气沉沉的士兵在每一章，让他越来越活跃。

# Direct3D入门

本章介绍了在学习本书其余部分之前需要了解的内容，换句话说......基础知识。我将快速浏览一下你应该知道的事情（设置Direct3D，windows循环等等）。如果本章所涉及的任何概念都没有掌握，那么您需要休息一下，重新温习一下这些概念。如前所述，您需要熟悉面向对象的C ++和DirectX 9。所以，在你处理这本书中更有趣的主题之前，让我们先看看基础知识。在本章中，我将介绍应用程序框架，创建窗口，设置Direct3D设备等。但是，由于这些主题并不真正属于本书的核心，我将简单介绍它们并向您展示启动和运行所需的最少代码量。但是要特别注意应用程序框架，因为这是构建本书中所有其他示例的框架类。 本章包括以下内容：

入门

应用框架

使用Direct3D进行渲染

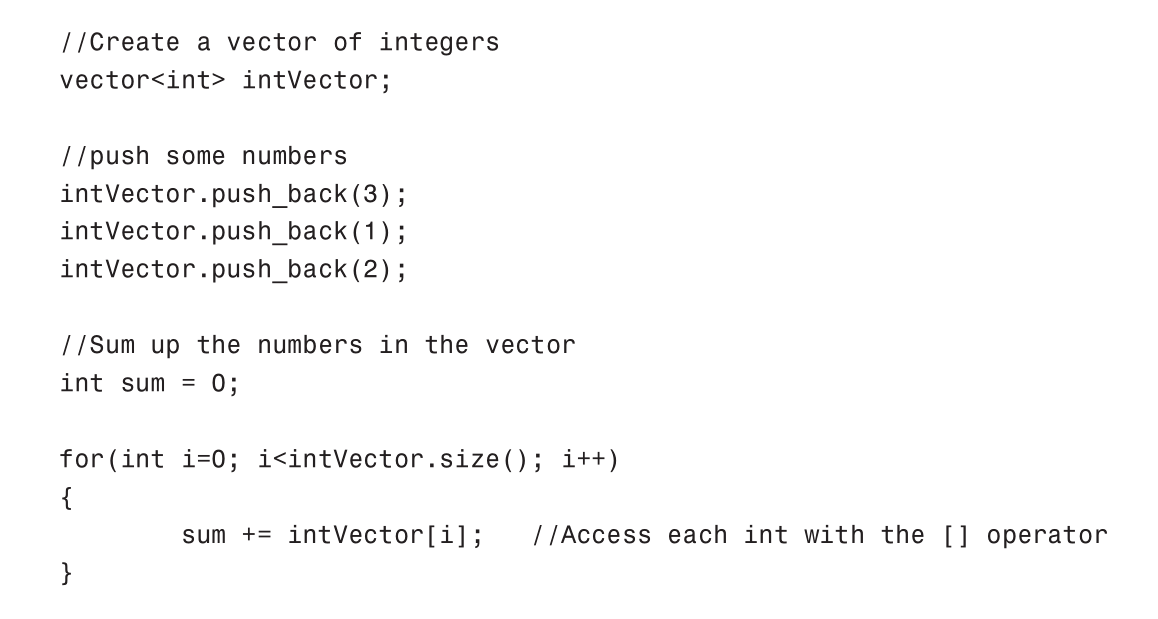
(请注意，本书中的所有代码都是在头脑中清晰地编写的，而不是后续修订。另外，为了保持简洁，不做错误检查。例如，我很少检查Direct3D/D3DX函数的返回值，而是简单地假设它们已经成功完成。同样，我假设有足够的内存来创建新的类、网格、纹理等等。因此，如果您打算在自己的项目中使用本书中的代码，请注意这个事实。)

在这本书中，我将使用DirectX 9来做所有的渲染和资源管理。你可能会问，为什么?新的DirectX 10已经出来了! 说实话，在DirectX 10中，覆盖相同主题所需的额外工作和支持代码数量实在是太大了只会使得尝试过于繁重。在DX9中，D3DX库中存在很多支持代码，其中大部分已经在DX10中被弃用（令我们的业余爱好程序员失望）。本书所需的最大缺失是加载.x文件（不是电视剧，而是DX9用来存储模型的文件格式）。在撰写本书时，使用DX10仍然没有任何简单的方法。 无论如何，如果你可以编写自己的网格导入器，你可能不需要我的帮助来移植本书中的例子，如果没有，那么，DX9是你最好的选择。

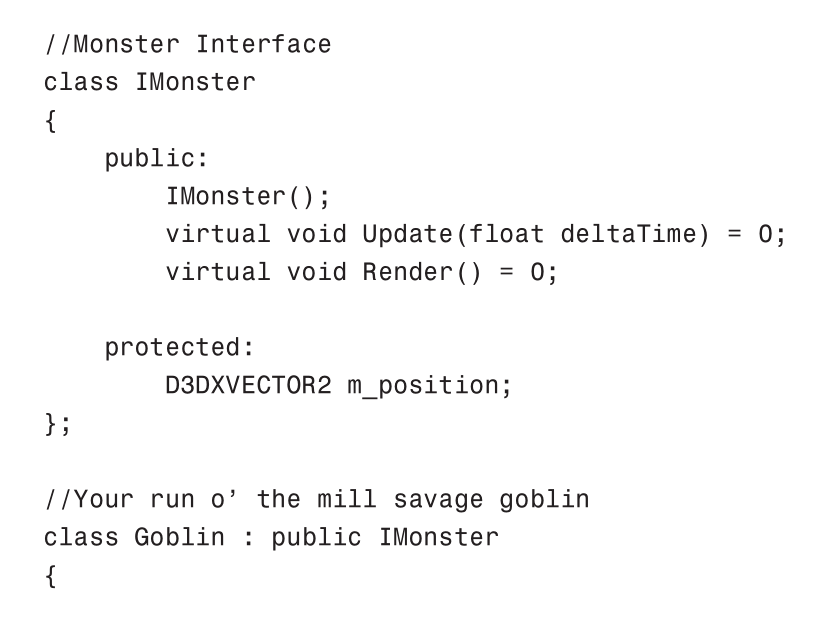
坚持使用DX9的另一个原因是，那里的大多数计算机仍然没有兼容DX10的显卡，而且可能至少在未来几年内也不会。

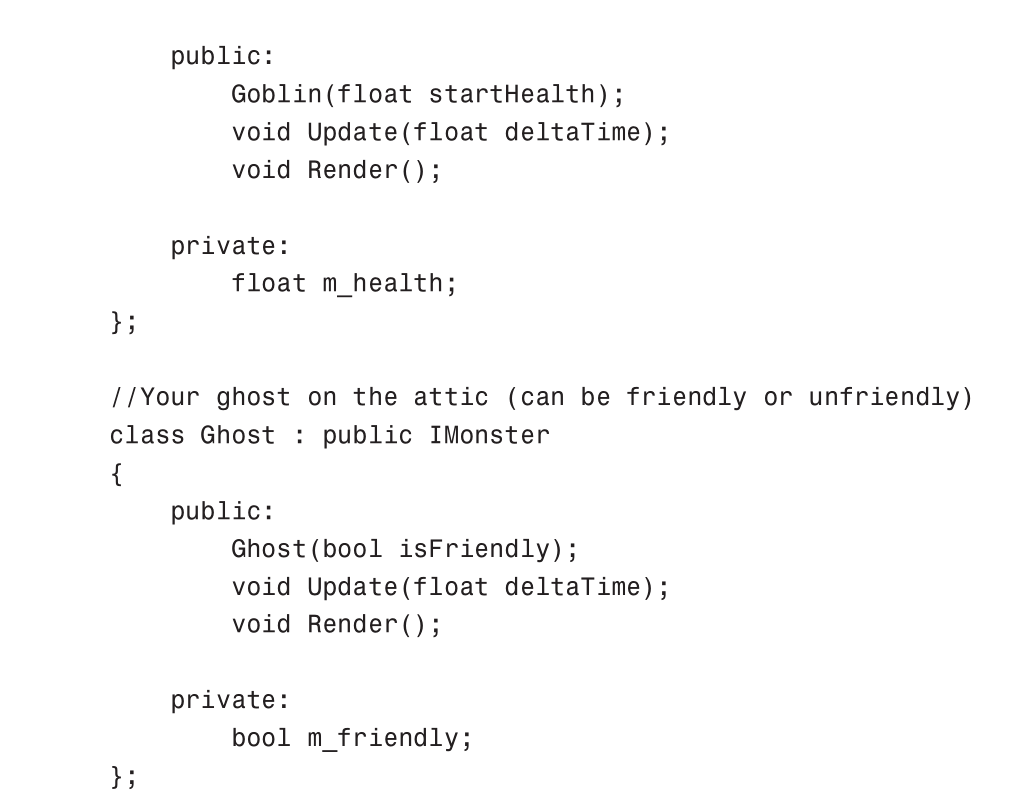
但是，无论您使用哪种版本的DirectX进行渲染，您仍将从本书中的课程中受益。本书中介绍的类和结构与DX9无特殊关系，可以轻松移植到其他渲染系统，如DX10甚至OpenGL。

重新发明轮子是我非常喜欢做的事情。不过，我就饶了你吧，因为你可能不会有同样的癖好。因此，我将严重依赖标准模板库（STL）用于我的所有数据容器类和Direct3D eXtension（D3DX）库所使用的数学函数，资源加载函数等。这里是stl :: vector类的简单使用， 如果你以前没见过：

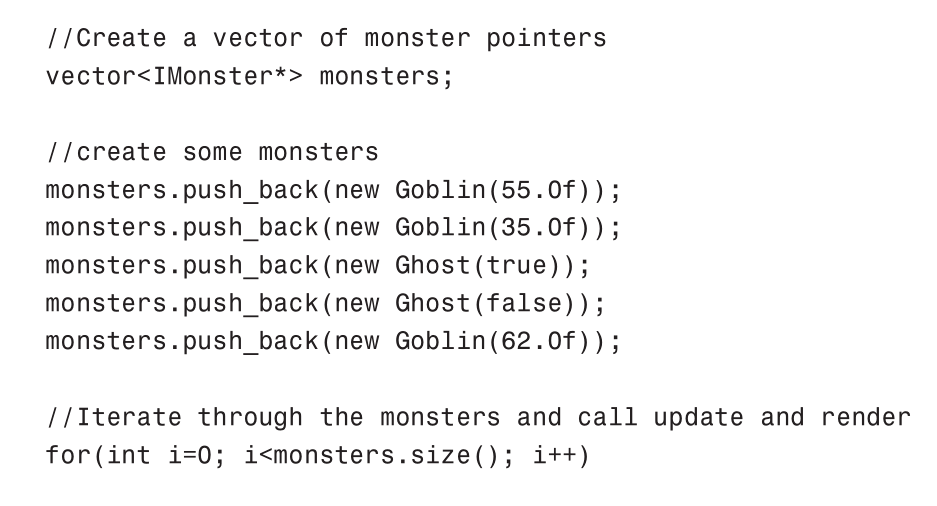


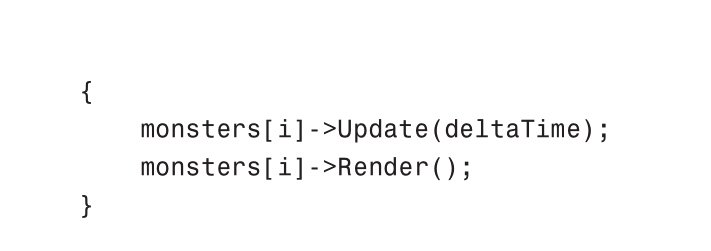
这是一个非常简单的例子，演示了一个整数对象数组。然而，在实践中它不是很有用;更常见的情况是，您有一个指向用户定义类的指针对象数组。看看下面三个类:





Ghost和Goblin类都继承自IMonster接口。 函数Update（）和Render（）在IMonster类中声明为纯虚函数，必须在从它继承的类中实现。但你应该已经知道这些了。如果没有，您可能需要温习一些基本的面向对象概念，如继承、多态性等。无论如何，回到STL数组，这里是你如何创建，更新和渲染一堆怪物：





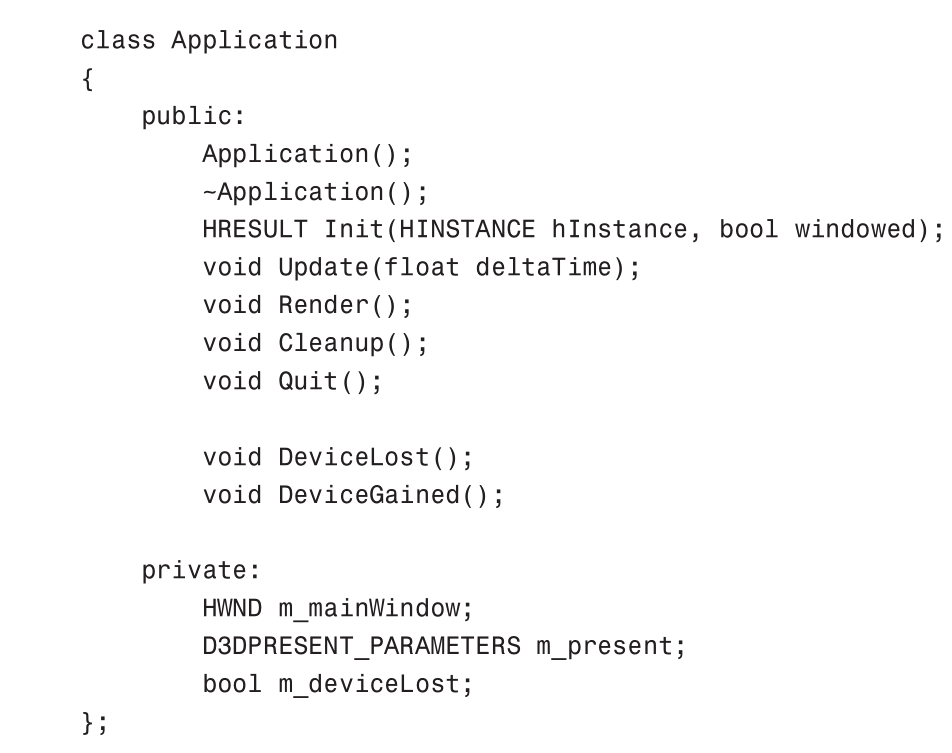
Ghost和Goblin可以具有完全不同的更新和渲染功能。 但是，由于它们都实现了IMonster接口，因此怪物数组并不关心数组中每个项目所代表的确切类。 在本书中，您将看到许多STL容器以类似的方式使用（数组，队列，映射等）。

这几乎涵盖了我将如何使用STL库。 另一方面，D3DX库是函数，结构和类的集合，在本书中也将广泛使用。您可以通过前缀（是的，您猜对了）D3DX识别D3DX功能等，如下所示：D3DXMATRIX，D3DXVECTOR3，D3DXVec3Normalize（），D3DXMatrixIdentity（）等等。我将尝试介绍书中使用的所有这些新功能（而不是在这里全部介绍）。 请记住，您有DirectX SDK文档，其中详细介绍了所有这些功能和结构。

现在使用Visual Studio Express 2008免费制作DirectX应用程序非常简单。如果您有其他版本的Visual Studio，设置DirectX项目的步骤是相同的。但请注意，早期版本的Visual Studio Express（如2005等）无法构建Win32应用程序而不会有一些额外的麻烦。 因此，除非您拥有Visual Studio的副本，否则最好坚持使用VS Express 2008。

接下来是DirectX的安装就不作介绍了。/\*\*\*\*百度都有。\*\*\*\*\*\*\*/

由于您将编写Win32应用程序来运行游戏，因此您需要一个类来处理主程序循环，图形设备的初始化等等。为此，我将使用Application类。 再一次，请记住，我是第2章Direct3D入门19图2.4项目属性页。 20使用Direct3D进行角色动画在此类中进行最少的错误检查以保持轻量级，因此对于任何“高级”应用程序，我建议将DXUT框架作为起点（随附DirectX SDK）。 记住这一点，这里是Application类的概述：

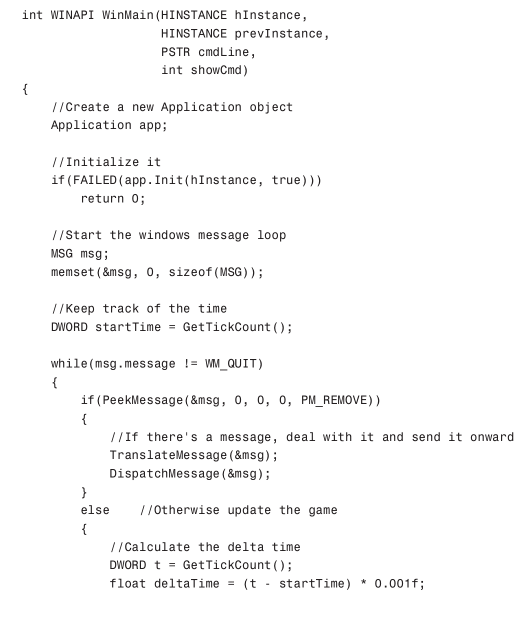


总而言之，这是一个简约的类。 函数的名称几乎可以解释它们的作用。 创建Application类之后，必须调用Init（）函数。 在此功能中，将加载资源并创建图形设备。调用Update()函数填输入变量为每一帧到此前一帧的时间差，之后再调用Render()函数。Update（）函数负责更新世界（即游戏），移动对象，更新物理引擎等。完成后，Render（）函数渲染所有对象并将结果呈现给屏幕。

DeviceLost（）和DeviceGained（）函数需要一些解释。 当设备丢失或获得时，将调用这些函数。当窗口调整大小或用户从全屏切换到窗口模式等时会发生这种情况。存储在视频存储器中的所有资源都需要在设备丢失事件上释放，并在重新获得设备时重新创建。（这也是你希望已经知道的东西以及本书不会涉及的内容。）

一旦应用程序运行其路线并调用Quit（）函数（通过在示例中按Esc或Alt + F4完成），将调用Cleanup（）函数，并在其中释放所有已创建的资源。那么Application类是如何使用的呢？

WinMain（）函数是Win32程序的入口点。 正是在这个功能中，整个应用程序都存在并运行。下面的代码显示了您将在下面的示例中使用的简单WinMain()函数:



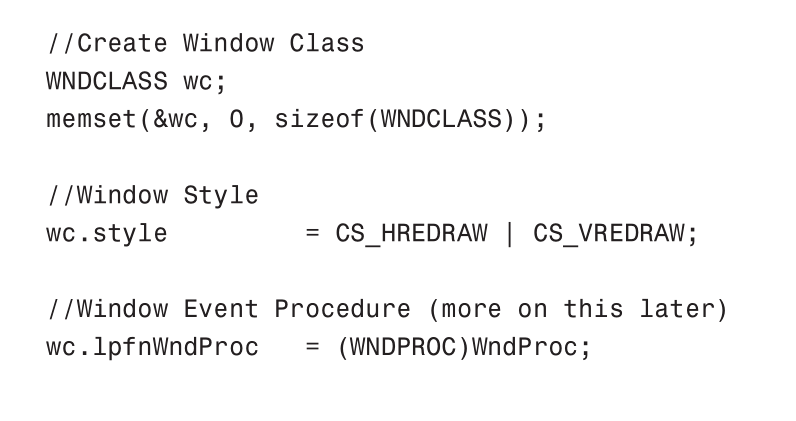


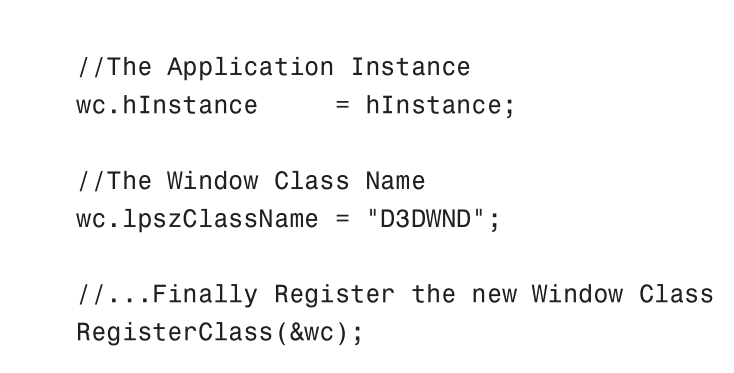
如您所见，我在WinMain（）函数的开头创建了一个Application类的实例。只要消息循环运行（没有收到WM\_QUIT消息），就会更新并渲染每个帧的Application实例。最后，调用Cleanup（）函数，在WinMain（）函数返回并且程序终止之前释放应用程序绑定的所有资源。

接下来让我们看看Application类的Init（）函数中发生了什么！

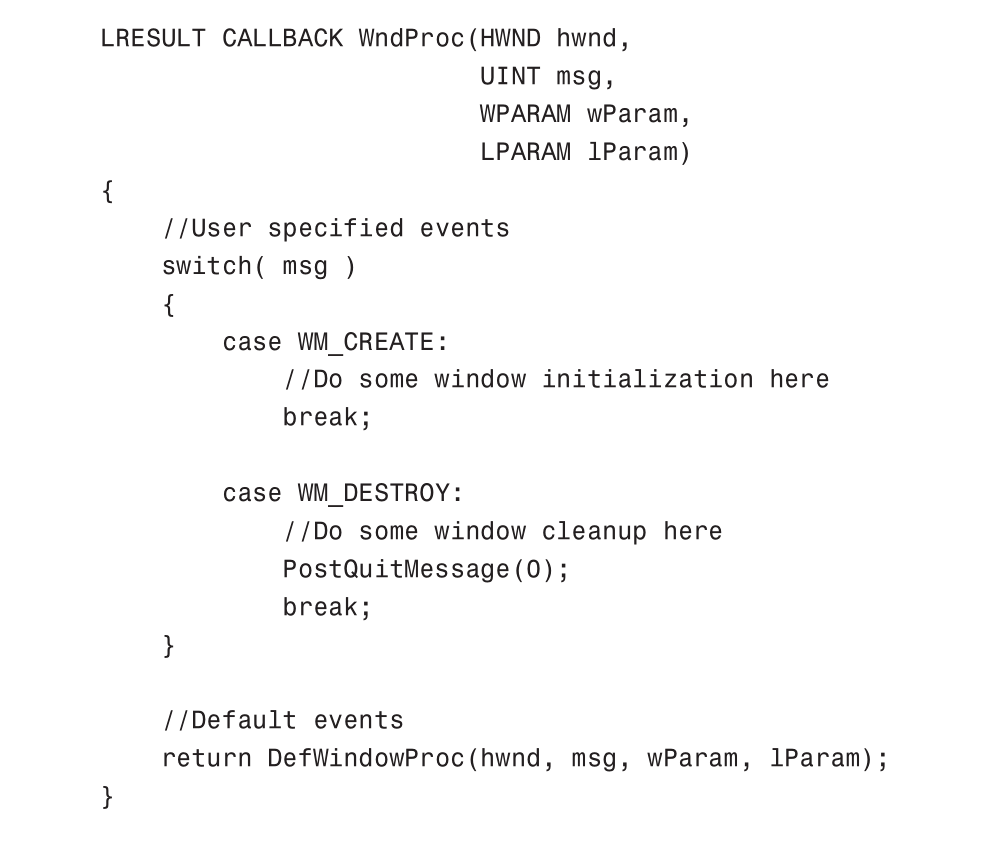
为了向用户显示3D世界，首先需要创建一个窗口。此窗口可以像在Windows下运行的任何其他窗口一样工作; 你可以调整它，最小化，最大化等。在这些例子中，我只会使用窗口来显示渲染后的3D世界的每一帧。

因此，要创建一个窗口，首先需要通过填写WNDCLASS结构来创建和注册窗口类，如下所示：

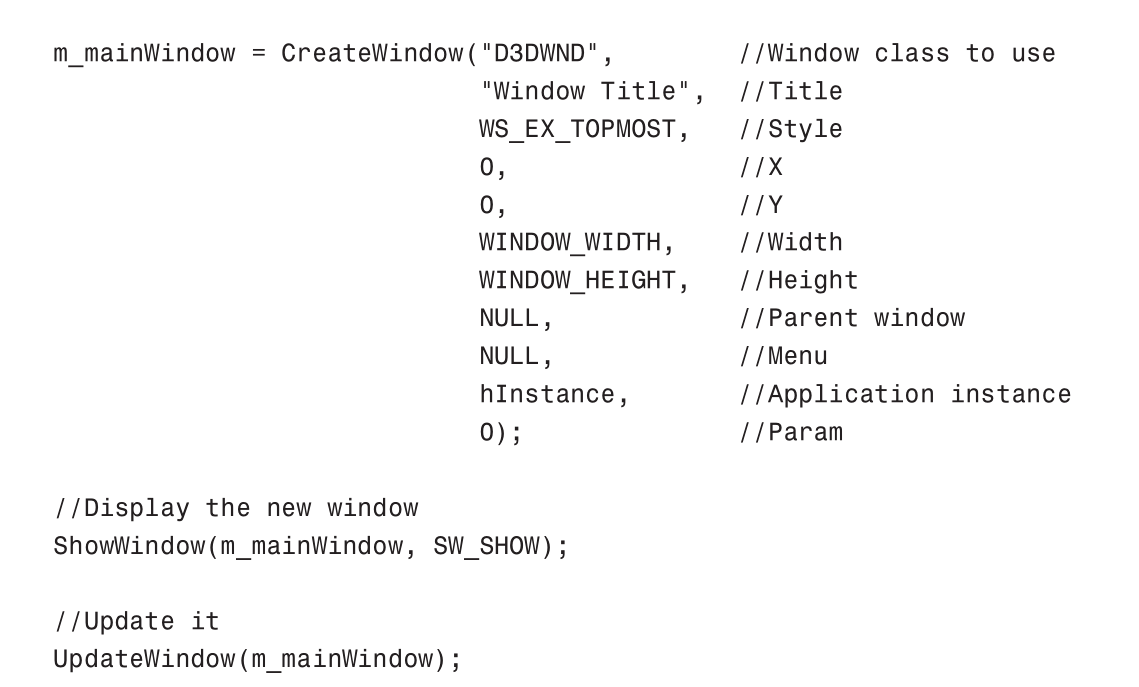




注册窗口类当然有很多选项可用。 这里的代码只显示了启动和运行所需的最少代码。 查看Microsoft Developer Network（MSDN）以获取有关如何创建窗口等的更多信息。我需要解释的一件事是lpfnWndProc变量。 lpfn前缀代表长指针函数，或者换句话说是函数指针。窗口过程处理窗口的所有传入事件，用户可以指定每个事件应该发生的事情。 此示例中使用的轻量级窗口过程如下所示：



在此窗口过程中仅处理WM\_CREATE和WM\_DESTROY函数。（有关您可以在窗口过程函数中捕获和处理的其他事件，请参阅MSDN。）然后我返回DefWindowProc（）函数的结果，第2章一个Direct3D Primer 23，它基本上是所有窗口事件的默认过程。注册窗口类并为其指定窗口过程后，可以使用CreateWindow（）函数创建此窗口类型的实例：



这里的所有都是它的。 您现在有一个窗口正在运行并正在更新。 Web上有大量关于Win32应用程序编程的资源和教程。对于本书，如果您对如何创建窗口，Windows主循环和窗口过程等有一般性的了解就足够了。这就很好地处理了windows代码...接下来，如何设置Direct3D！

在本节中，我将介绍如何设置Direct3D设备并将某些内容绘制到屏幕上。Direct3D设备是您将用于将对象绘制到屏幕、创建资源等等的接口。由于我确实希望您在处理本书之前有一些使用Direct3D的经验，因此我将简要介绍本节。 如果不清楚，请参考DirectX SDK文档或许多可用的入门书籍之一。有关Direct3D游戏编程的入门书籍，我推荐Frank Luna使用Direct X 9.0c进行3D游戏编程简介：《A Shader Approach》。

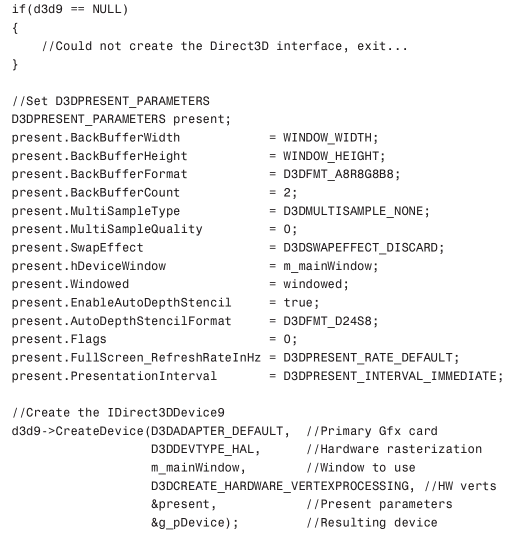
通过以下步骤初始化Direct3D设备：

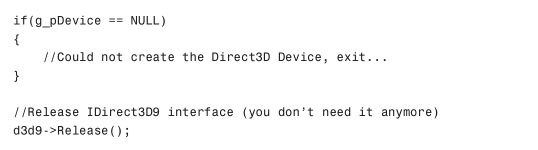
1.创建Direct3D接口。

2.填写D3DPRESENT\_PARAMETERS结构。

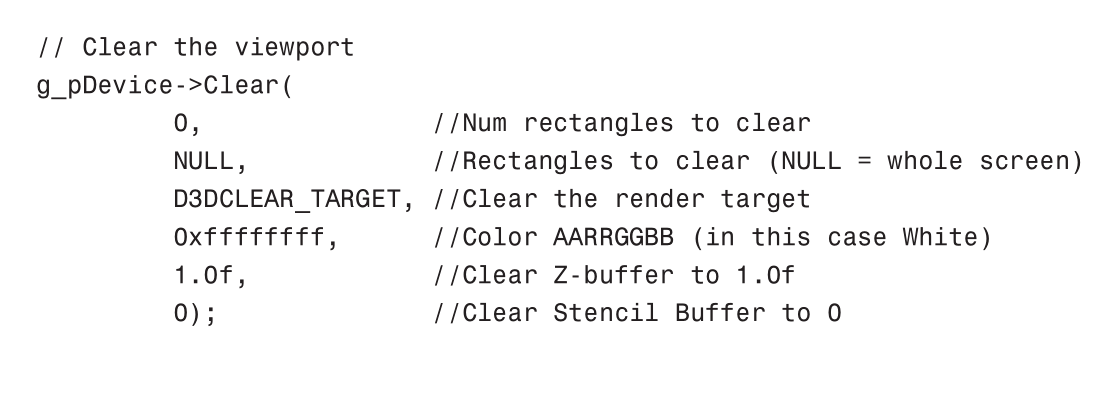
3.创建Direct3D设备。

以下是这些步骤的代码：

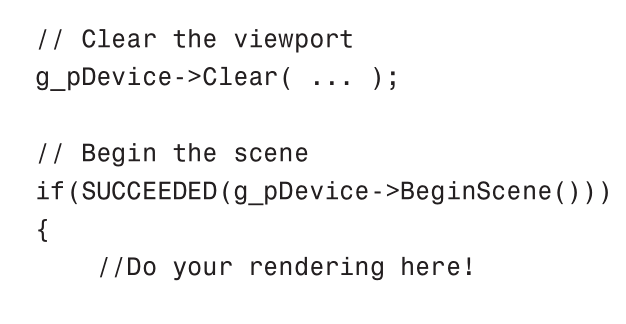


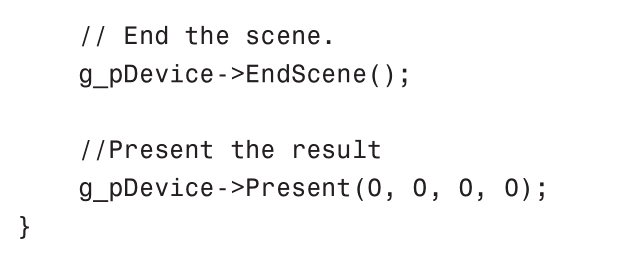


此代码将设置Direct3D设备，假设您有一个支持硬件光栅化，硬件顶点处理，所选后缓冲区格式等的图形卡。我还将完成的Device存储为全局指针，以便可以从Application类以外的类访问它。设备输出现在连接到之前创建的窗口。 因此，如果要将窗口的背景清除为某种颜色，可以使用以下代码：



Direct3D的渲染循环非常简单，由三个函数控制：BeginScene（），EndScene（）和Present（）。在BeginScene（）和EndScene（）函数之间可以进行渲染/绘制，完成后调用Present（）函数将结果显示在屏幕上。Present（）函数自动处理后台缓冲区交换等，因此您不必担心这一点。



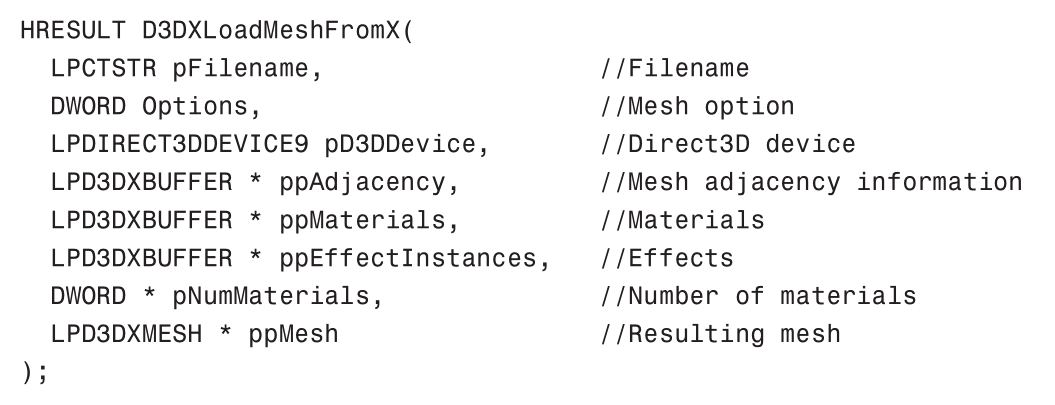


将零传递给所有Present（）函数参数会将结果显示在整个窗口全屏幕上，也就是说，这就是您想要的。 有关详细信息，请参阅DirectX SDK文档。这已经是很多初始化代码了。只要再和我呆一会儿，你就会在屏幕上看到一些东西。

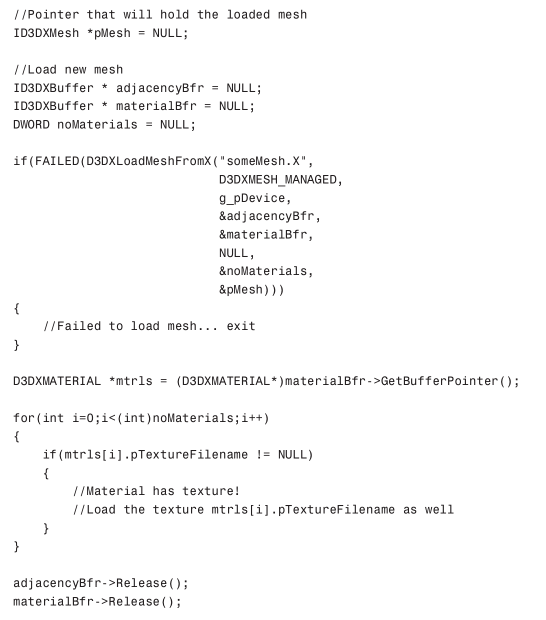
在本章中，我将只加载一个静态网格物体，这样就有些东西可以渲染到现在相当空白的屏幕。

网格是通过ID3DXMesh接口存储和访问的，在本书的最后您会对这个类更加熟悉。 现在，只要知道它包含网格或模型就足够了。

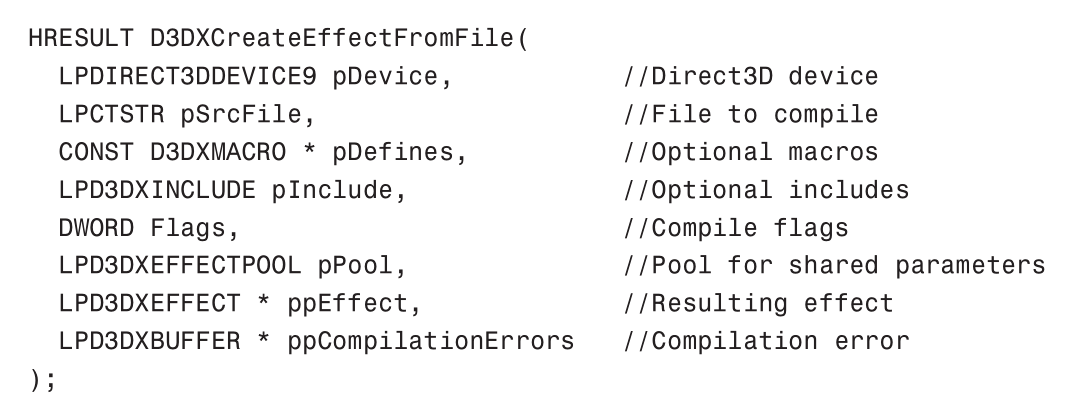
在本书中，我将使用.x格式以及D3DX库中可用的网格加载函数，并加载静态网格物体我将使用以下函数：



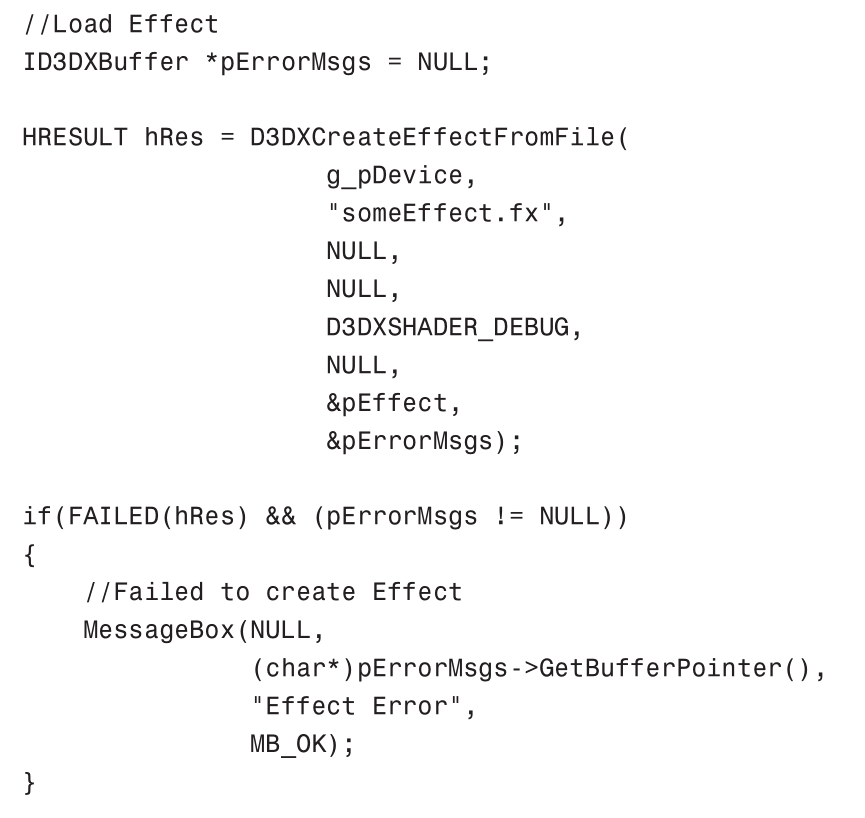
由于.x格式还可以包含嵌入的材质，甚至着色效果，所以D3DXLoadMeshFromX()函数也有返回这些参数的参数。



您将在本章的下一个示例中找到加载和存储网格的完整代码。现在你有了“渲染的东西”，接下来您需要整理出“如何渲染”。今天，这是通过顶点和像素着色器完成的。我假设您对像素和顶点着色器的工作原理有所了解。如何利用指令集合的Effect文件包括顶点和像素着色器来渲染特定效果。在高级别上，效果文件可以包含一个或28个带有Direct3D更多技术的角色动画，每个技术可以包含一个或多个过程（某些效果需要多次渲染几何体）。Effect文件由高级着色语言（HLSL）组成，我将在本书中使用它来创建一些更高级的效果。 以下函数在运行时加载并编译来自硬盘驱动器的Effect文件：

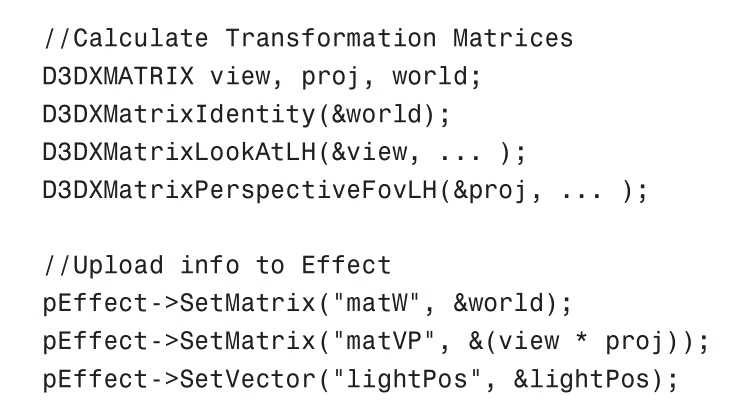


这是使用D3DXCreateEffectFromFile（）函数的代码：

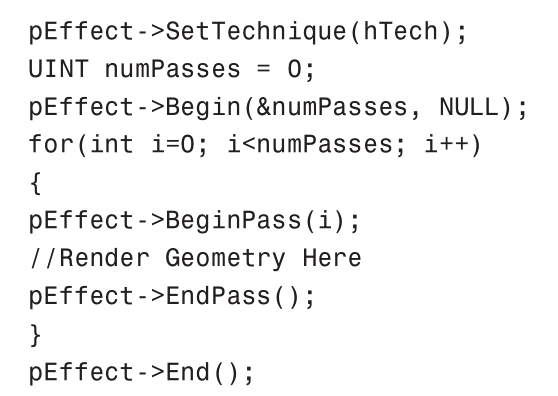


现在效果已经加载、编译(希望没有错误)并存储在ID3DXEffect接口中。在本书中，我将不讨论HLSL的语法(因为这本身就需要一本书)。只要说HLSL与C语法非常接近就足够了，即使您是新手，理解它也应该没有问题。

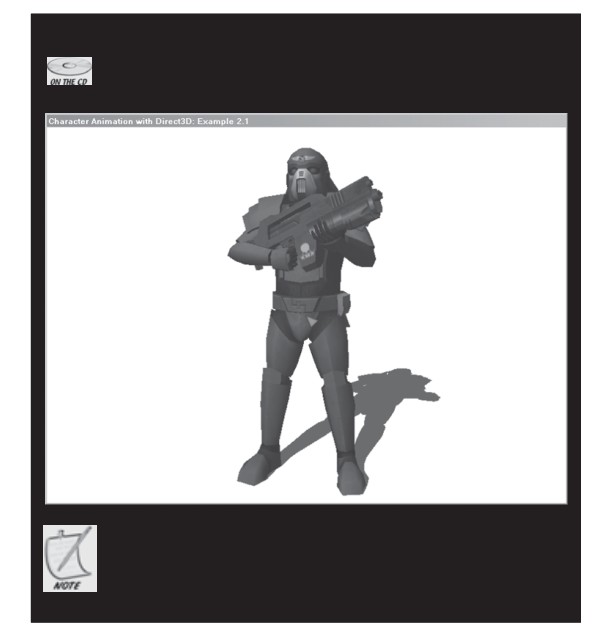
最后，在这里您将看到屏幕上出现的内容。一旦创建了转换矩阵（世界，视图和投影矩阵），您需要将这些（以及您需要的任何其他信息）上传到Effect。这样做：



这会将矩阵上传到效果。您可以分别使用SetVector（）和SetFloat（）函数上传向量，浮点数等。 一旦上传了Effect需要渲染的所有信息，就可以进行实际渲染。



去吧 加载的网格现在将呈现在屏幕上。



在本章中，我介绍了创建和运行Direct3D应用程序所需的所有必要的基础工作。我承认，这可能是有史以来对这个主题最简短的介绍之一。然而，如果本章所涉及的任何主题让您觉得不熟悉，我建议您在继续之前仔细阅读这些主题。本书未涉及的另一件事是高级着色语言（HLSL），这是您在继续之前至少需要了解的内容。

希望您还在阅读，并且没有被本章所涵盖的所有基础代码所影响。从现在起直到书的结尾，摆在台面上的除了人物动画其它什么都没有。