# 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

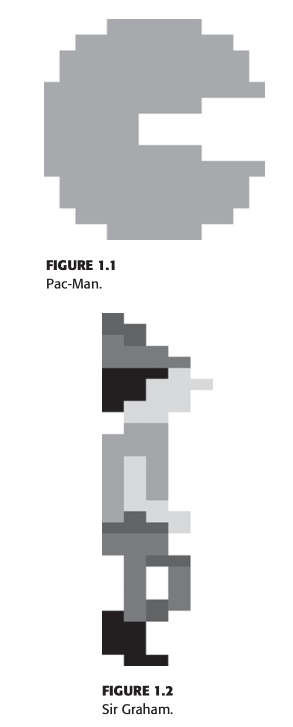
动画制作，我想游戏制作也属于这一类。然而，我可能会试图在“让角色以真实的方式移动“的方向上定义它。虽然，我认为这更好地回答了这个问题-“什么是角色动画的目标”。

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。

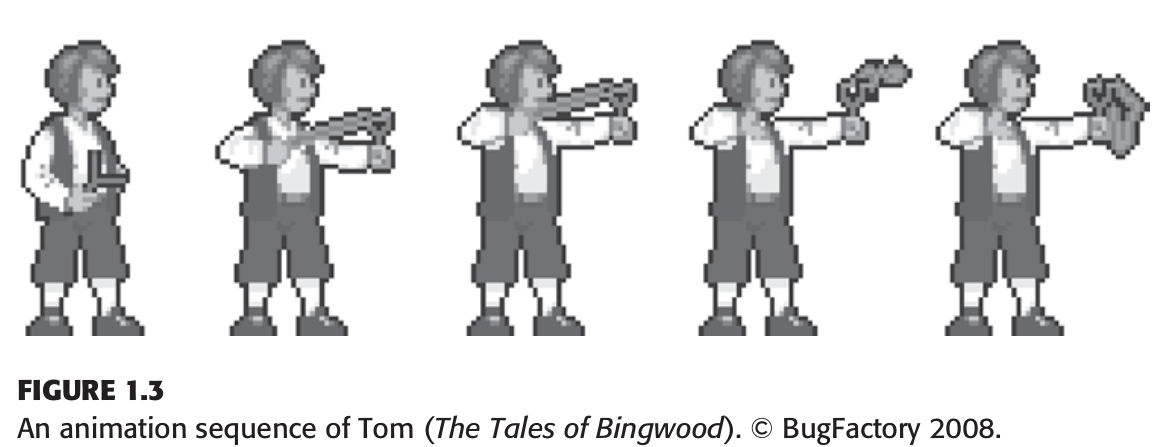
每年都有新的动画角色模型出现，推动了该领域的发展。本书所涉及的技术绝不是最前沿的。相反，它们是所有(或大多数)前沿技术的基础技术。骨骼动画、变形、布偶物理仿真、逆运动学等技术已经存在了很长时间(至少在游戏革新方面)。

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。

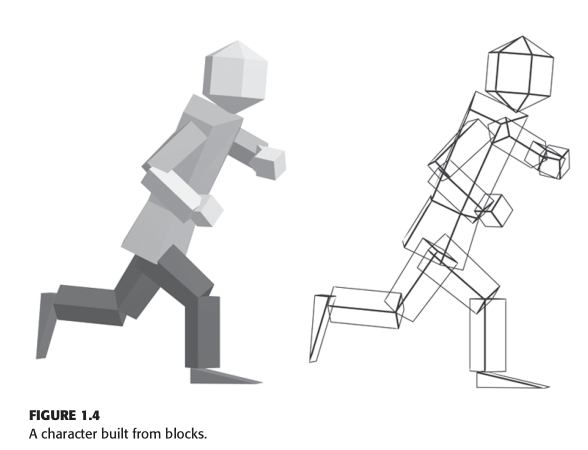


90年代，随着新十年的到来，第一批3D游戏带来了一系列全新的问题。一些更著名的游戏包括《沃尔芬斯坦3D版》(Wolfenstein 3Dand)，后来的《末日》(doomand)——尽管这些游戏并不能真正被称为真正的3D游戏，因为它们仍然使用2D精灵去创造敌人和角色(通常从8个角度绘制，这取决于它们面向玩家的方向)。第一个真正的3D角色出现在1992年发行的游戏《Alone in The Dark》中。它以全3D的角色为特色，带有插值动画。

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。

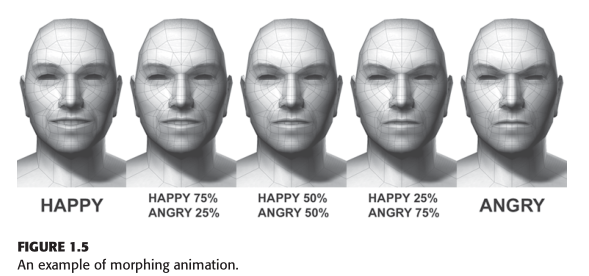
你可以很容易地在这个角色的关节间看到明显的间隙，但在那个时候通常没有模型的光照，而且分辨率非常小，以至于这些间隙通常对玩家隐藏。

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。



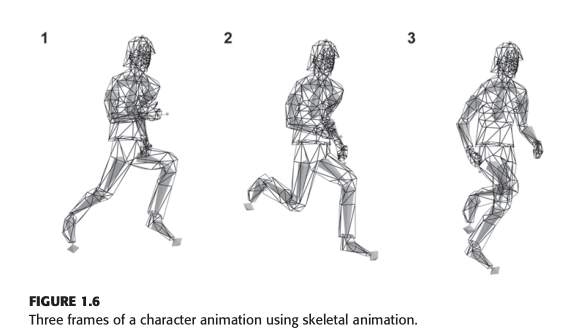
快速转发再次将我们带到了1998年，当时Sierra Studios发布了Half-Life（半条命~）游戏（由Valve开发）。Half-Life建立在Quake引擎的高度修改版本之上。最值得注意的是，游戏开发者添加了一个新的骨骼动画系统，允许他们在不同的角色上重复使用动画。

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）。

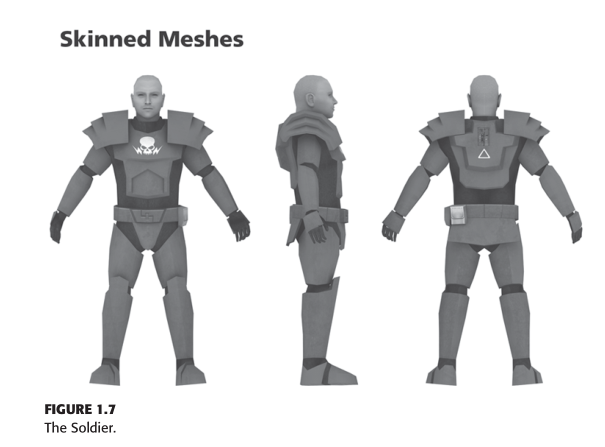
根据今天的标准，模型的复杂程度介于中低范围内：

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

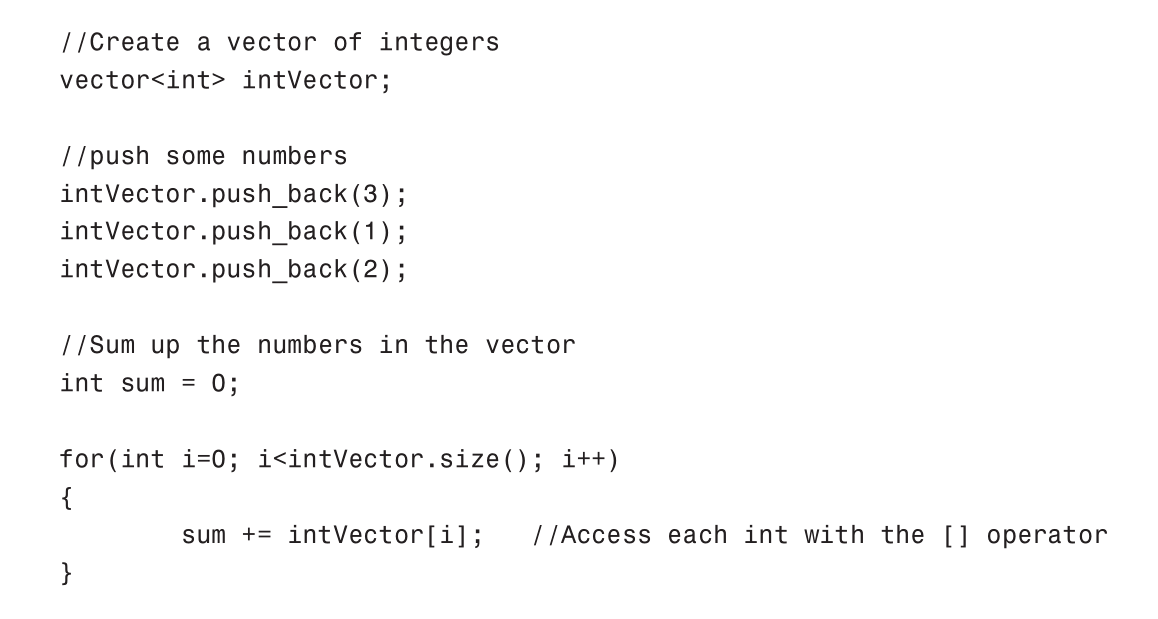
(请注意，本书中的所有代码都是在头脑中清晰地编写的，而不是后续修订。另外，为了保持简洁，不做错误检查。例如，我很少检查Direct3D/D3DX函数的返回值，而是简单地假设它们已经成功完成。同样，我假设有足够的内存来创建新的类、网格、纹理等等。因此，如果您打算在自己的项目中使用本书中的代码，请注意这个事实。)

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。

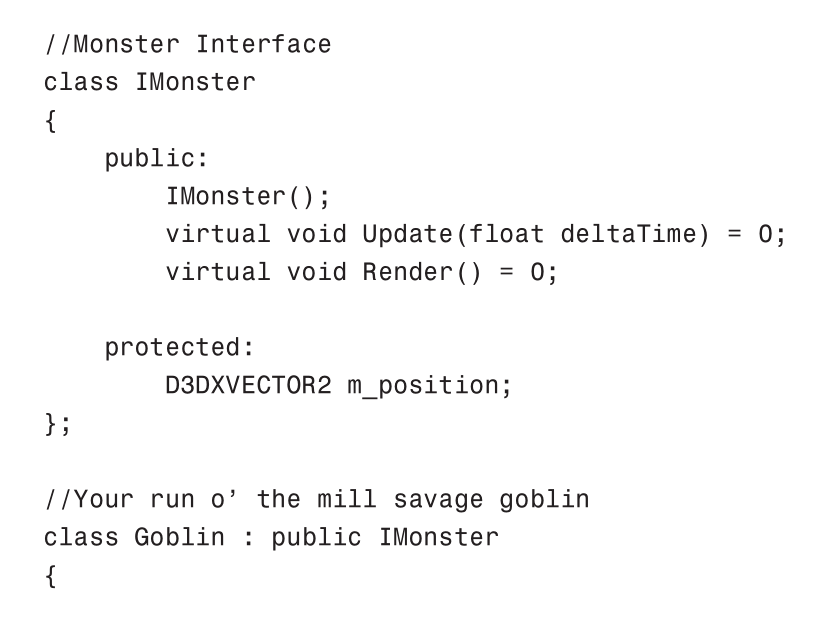
坚持使用DX9的另一个原因是，那里的大多数计算机仍然没有兼容DX10的显卡，而且可能至少在未来几年内也不会。

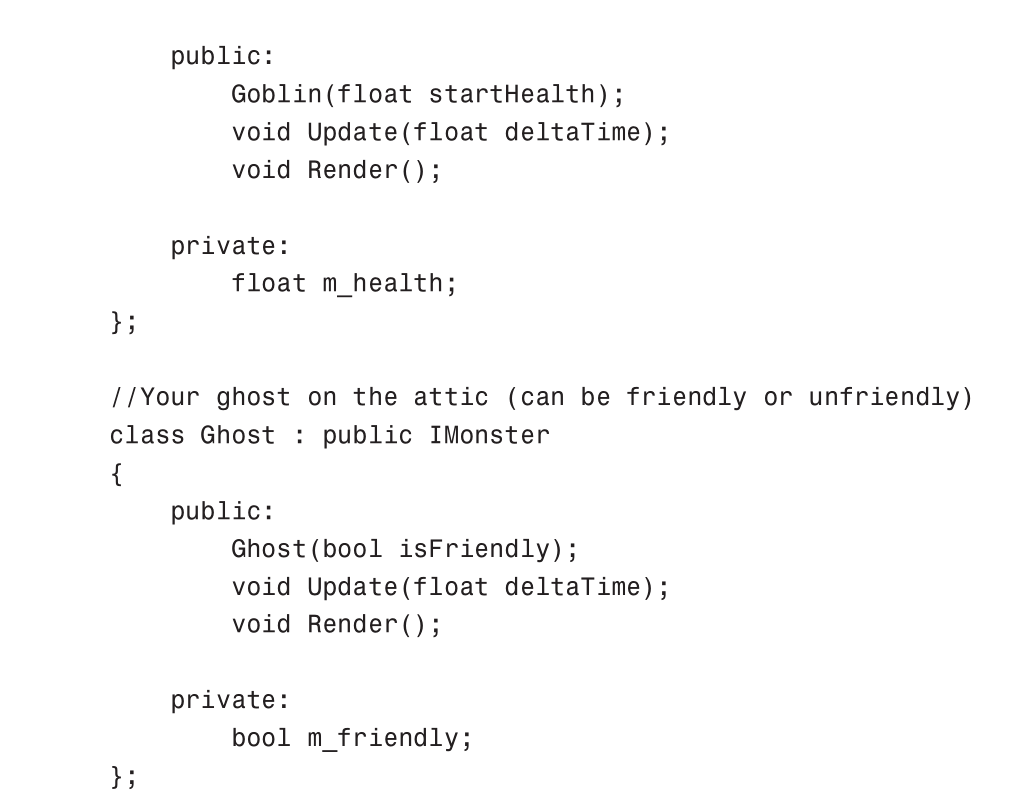
但是，无论您使用哪种版本的DirectX进行渲染，您仍将从本书中的课程中受益。本书中介绍的类和结构与DX9无特殊关系，可以轻松移植到其他渲染系统，如DX10甚至OpenGL。

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：

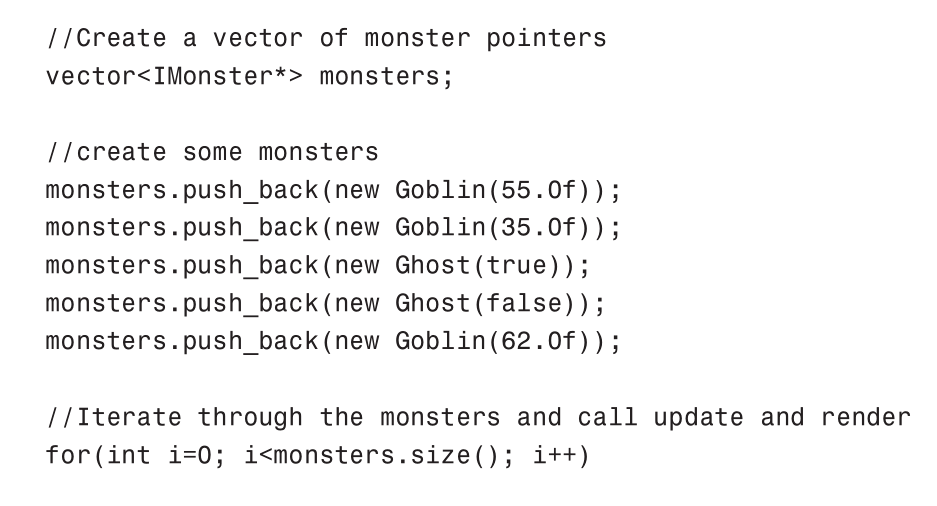


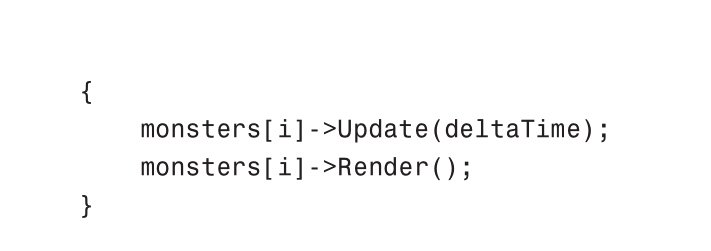
这是一个非常简单的例子，演示了一个整数对象数组。然而，在实践中它不是很有用;更常见的情况是，您有一个指向用户定义类的指针对象数组。看看下面三个类:





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：





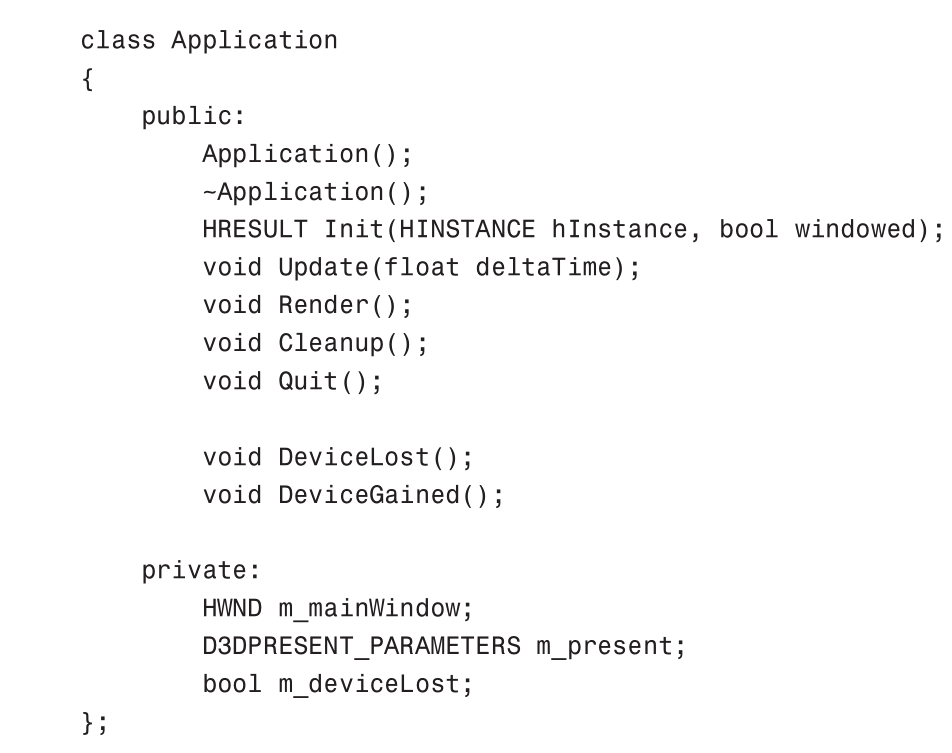
Ghost和Goblin可以具有完全不同的更新和渲染功能。但是，由于它们都实现了IMonster接口，因此怪物数组并不关心数组中每个项目所代表的确切类。在本书中，您将看到许多STL容器以类似的方式使用（数组，队列，映射等）。

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：

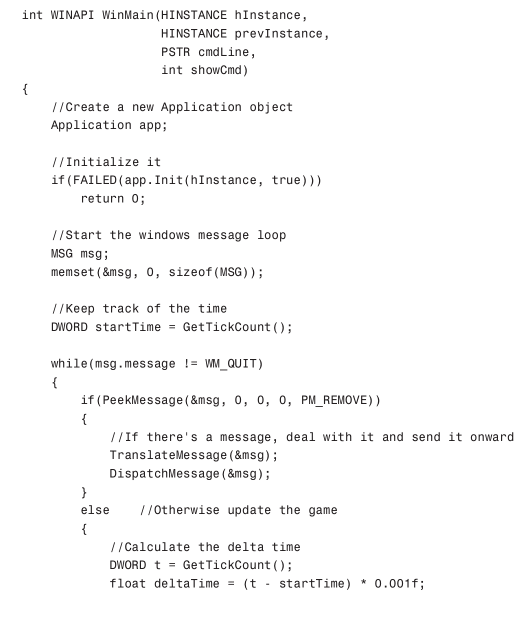


总而言之，这是一个简约的类。函数的名称几乎可以解释它们的作用。创建Application类之后，必须调用Init（）函数。在此功能中，将加载资源并创建图形设备。调用Update()函数填输入变量为每一帧到此前一帧的时间差，之后再调用Render()函数。Update（）函数负责更新世界（即游戏），移动对象，更新物理引擎等。完成后，Render（）函数渲染所有对象并将结果呈现给屏幕。

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:



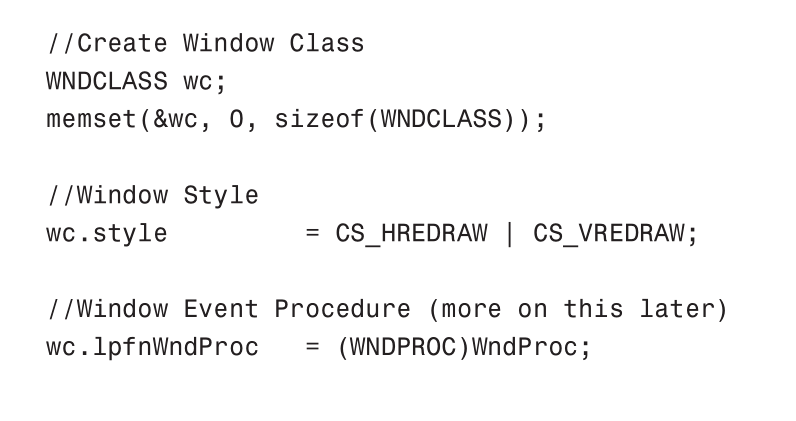


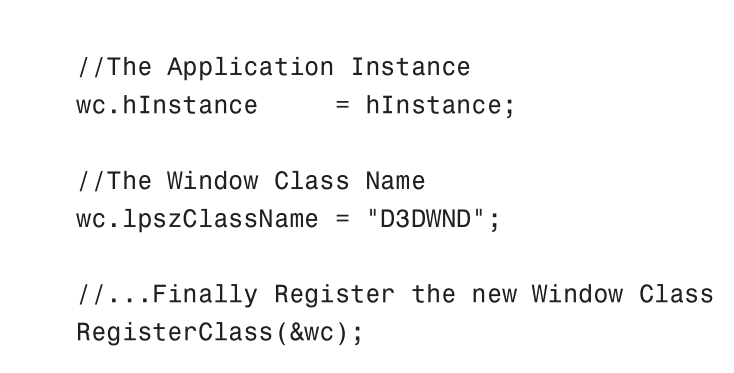
如您所见，我在WinMain（）函数的开头创建了一个Application类的实例。只要消息循环运行（没有收到WM\_QUIT消息），就会更新并渲染每个帧的Application实例。最后，调用Cleanup（）函数，在WinMain（）函数返回并且程序终止之前释放应用程序绑定的所有资源。

接下来让我们看看Application类的Init（）函数中发生了什么！

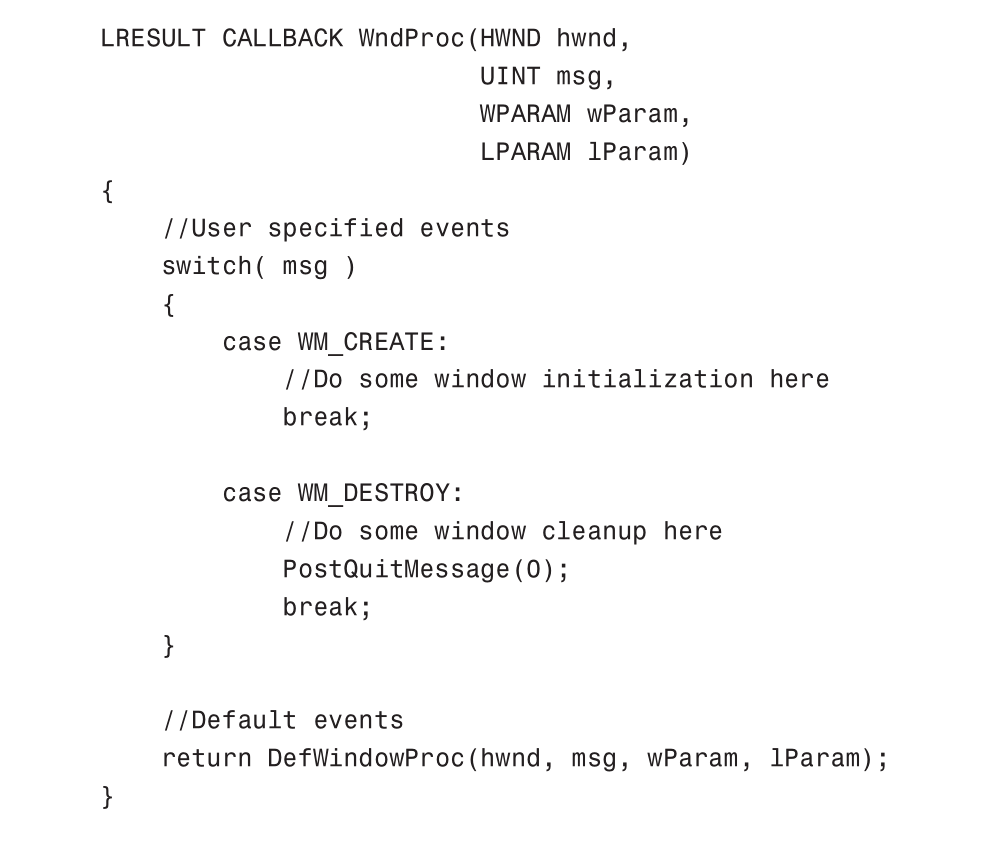
为了向用户显示3D世界，首先需要创建一个窗口。此窗口可以像在Windows下运行的任何其他窗口一样工作; 你可以调整它，最小化，最大化等。在这些例子中，我只会使用窗口来显示渲染后的3D世界的每一帧。

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：

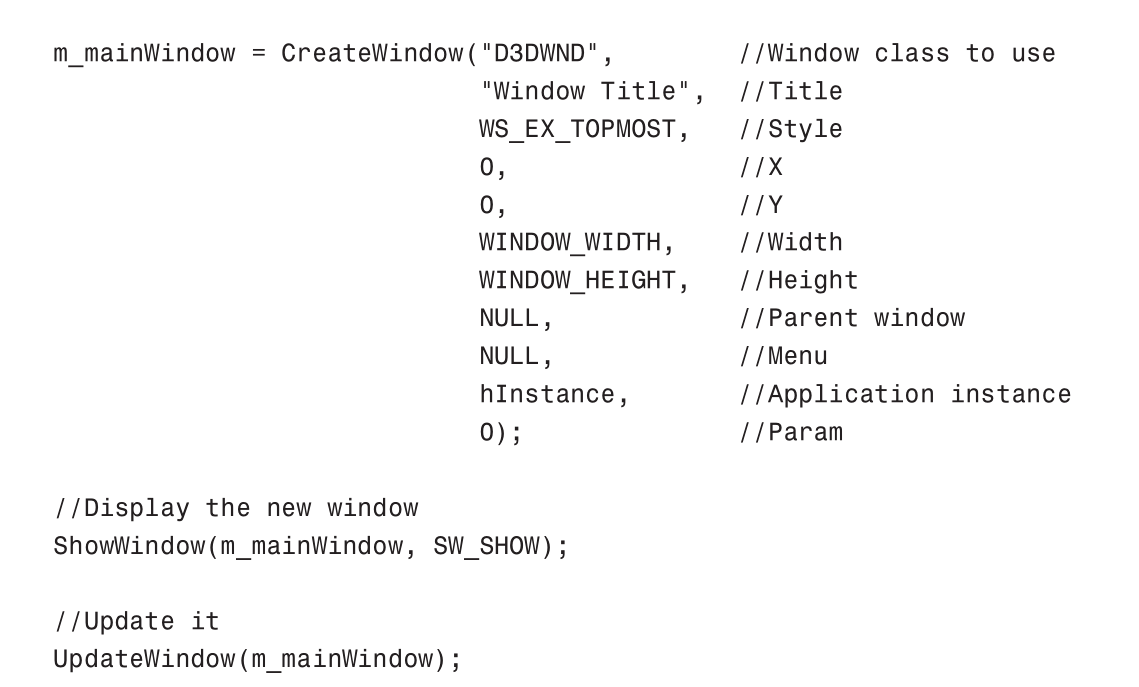




注册窗口类当然有很多选项可用。这里的代码只显示了启动和运行所需的最少代码。查看Microsoft Developer Network（MSDN）以获取有关如何创建窗口等的更多信息。我需要解释的一件事是lpfnWndProc变量。lpfn前缀代表长指针函数，或者换句话说是函数指针。窗口过程处理窗口的所有传入事件，用户可以指定每个事件应该发生的事情。此示例中使用的轻量级窗口过程如下所示：



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》。

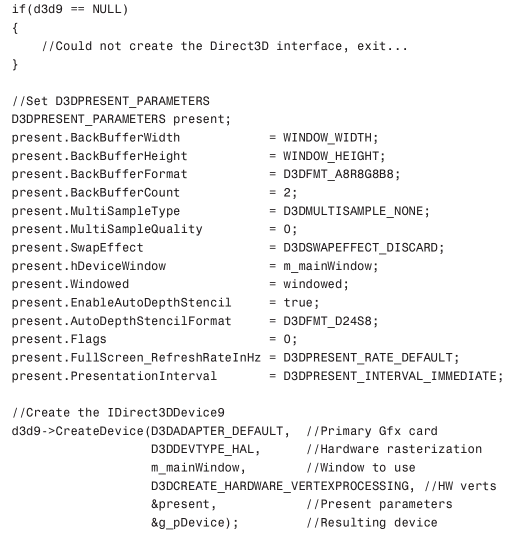
通过以下步骤初始化Direct3D设备：

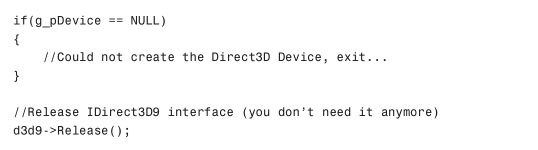
1.创建Direct3D接口。

2.填写D3DPRESENT\_PARAMETERS结构。

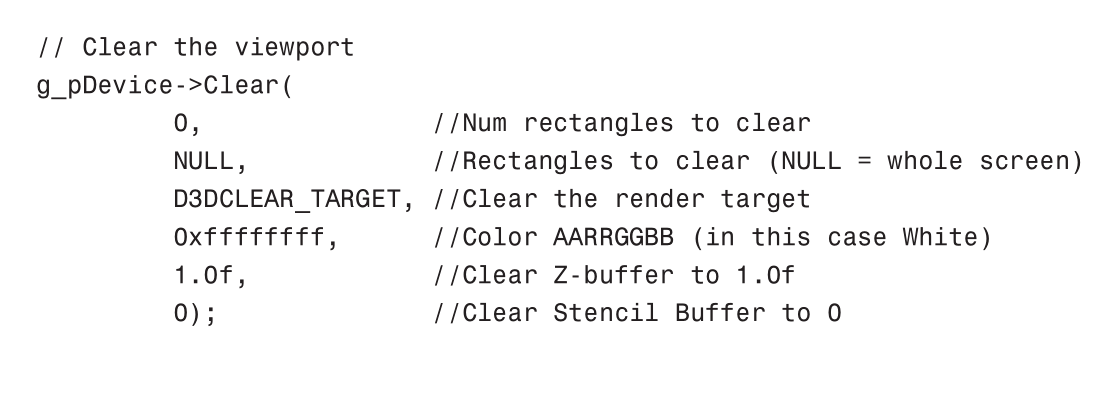
3.创建Direct3D设备。

以下是这些步骤的代码：

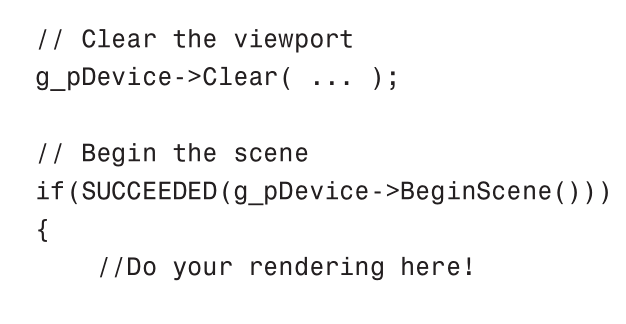


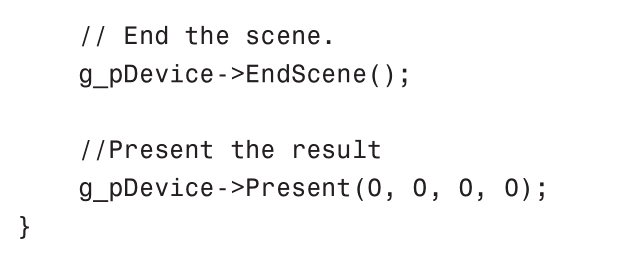


此代码将设置Direct3D设备，假设您有一个支持硬件光栅化，硬件顶点处理，所选后缓冲区格式等的图形卡。我还将完成的Device存储为全局指针，以便可以从Application类以外的类访问它。设备输出现在连接到之前创建的窗口。因此，如果要将窗口的背景清除为某种颜色，可以使用以下代码：



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。



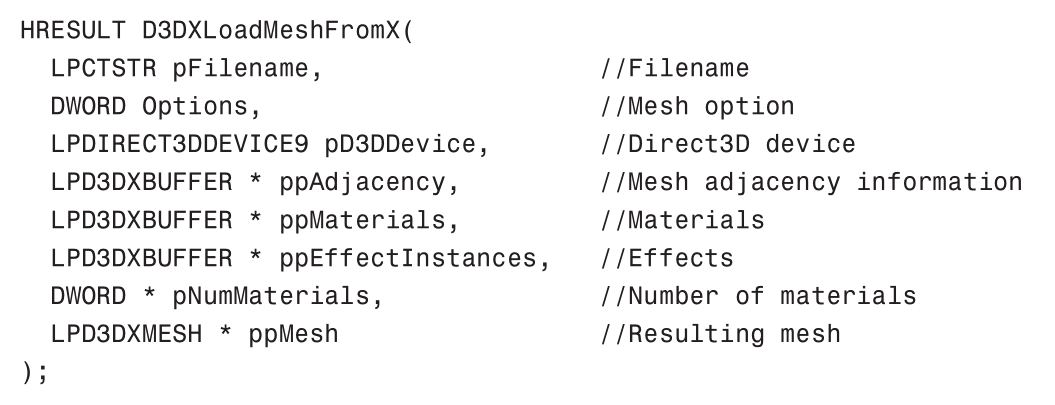


将零传递给所有Present（）函数参数会将结果显示在整个窗口全屏幕上，也就是说，这就是您想要的。有关详细信息，请参阅DirectX SDK文档。这已经是很多初始化代码了。只要再和我呆一会儿，你就会在屏幕上看到一些东西。

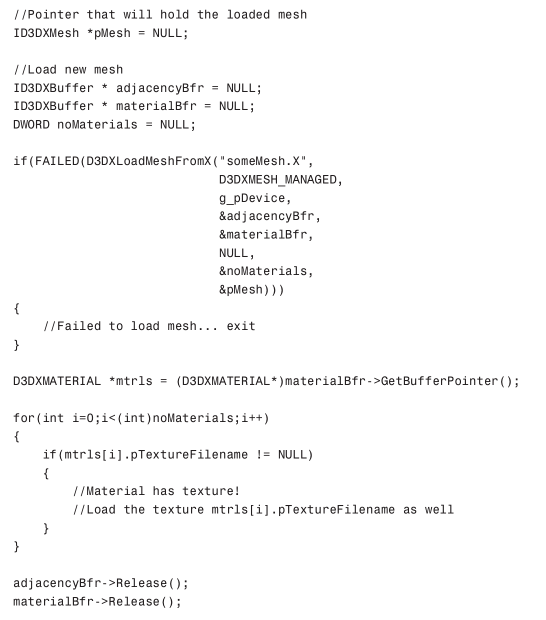
在本章中，我将只加载一个静态网格物体，这样就有些东西可以渲染到现在相当空白的屏幕。

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。

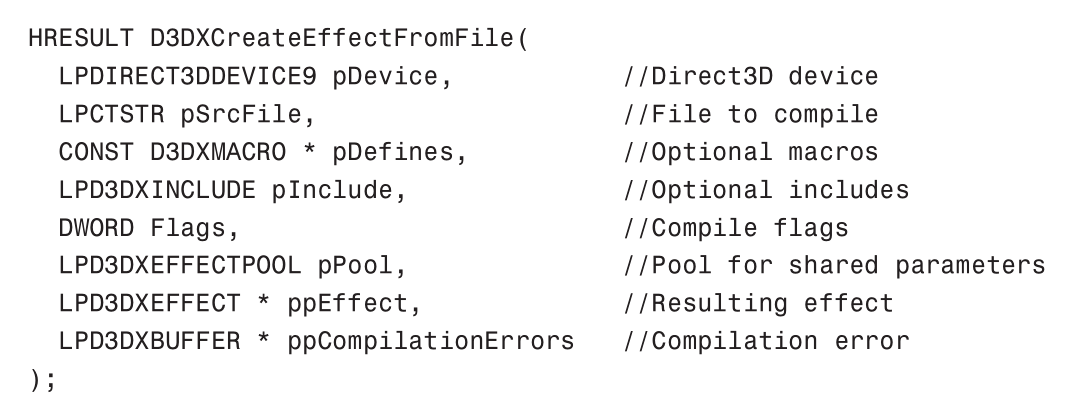
在本书中，我将使用.x格式以及D3DX库中可用的网格加载函数，并加载静态网格物体我将使用以下函数：



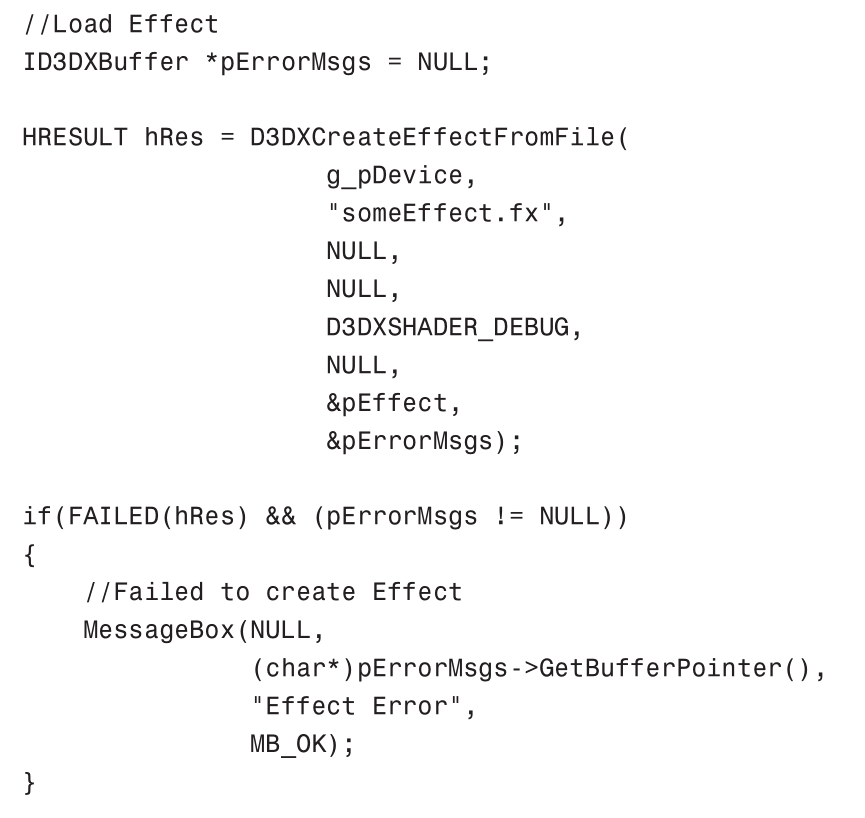
由于.x格式还可以包含嵌入的材质，甚至着色效果，所以D3DXLoadMeshFromX()函数也有返回这些参数的参数。



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：

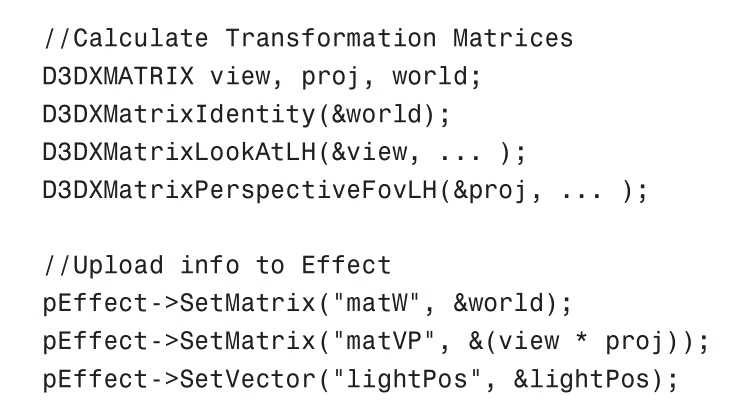


这是使用D3DXCreateEffectFromFile（）函数的代码：

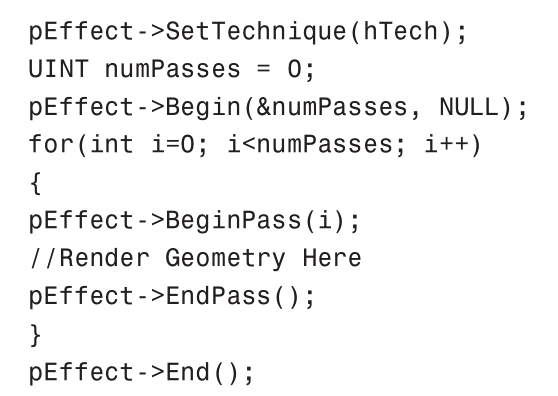


现在效果已经加载、编译(希望没有错误)并存储在ID3DXEffect接口中。在本书中，我将不讨论HLSL的语法(因为这本身就需要一本书)。只要说HLSL与C语法非常接近就足够了，即使您是新手，理解它也应该没有问题。

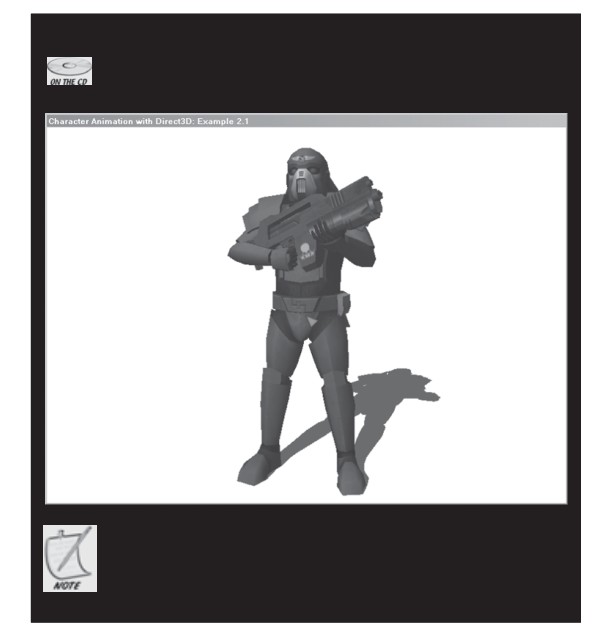
最后，在这里您将看到屏幕上出现的内容。一旦创建了转换矩阵（世界，视图和投影矩阵），您需要将这些（以及您需要的任何其他信息）上传到Effect。这样做：



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。



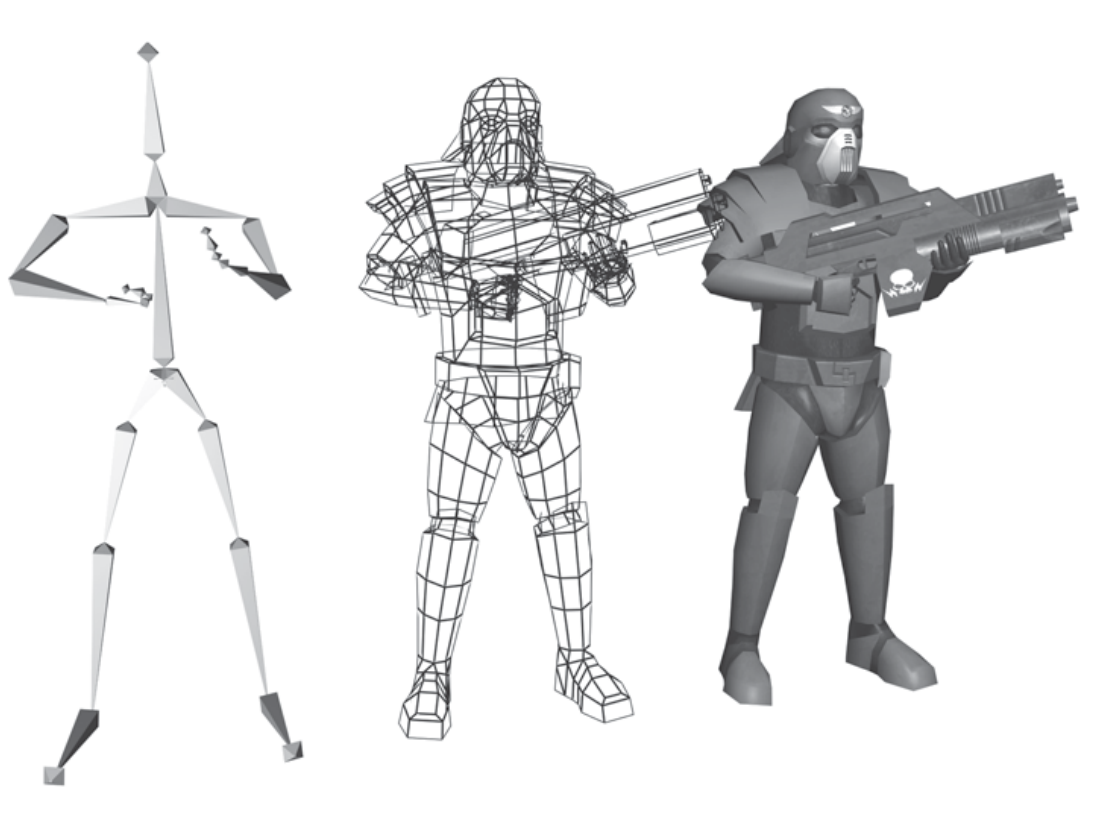
去吧 加载的网格现在将呈现在屏幕上。



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。

# 蒙皮网格



髋骨和后骨相连，

背骨和颈骨相连，

颈骨和头骨相连，

哦，天哪，他们多吓人啊! -Dem Dry Bones, traditional spiritual

在上一章中，您加载并渲染了一个静态网格物体（士兵模型）。但是，要创建一个可以制作动画的角色，首先必须给它一些骨骼。在本章中，您将学习骨骼结构的基础知识，更重要的是，您将学习如何将网格划分为这样的结构（称为蒙皮）。这种类型的结构是由以分层方式链接在一起的几个骨骼创建的。骨骼通常具有父骨骼和零个或多个子骨骼。应用于父骨骼的任何变换也会影响其子项（及其子项等）。构建或加载分层骨骼结构后，需要对其应用网格，以便网格中的每个顶点都链接到一个或多个骨骼。

如果您是第一次看这个主题，那么现在看起来有点令人困惑，但不要担心。一切都会变得更加清晰。在本章中，您将了解以下内容：

骨骼层次结构的基础知识

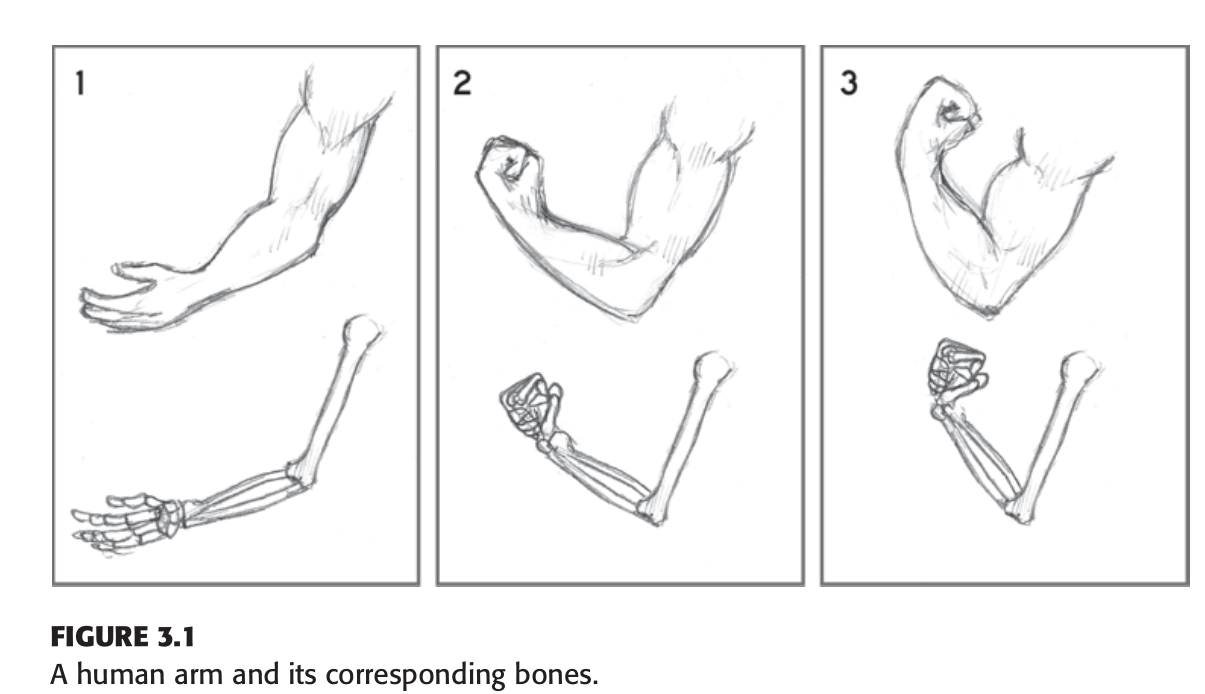
从.x文件加载骨骼层次结构

软件皮肤化

硬件皮肤化

在骨骼层次结构中渲染静态网格物体

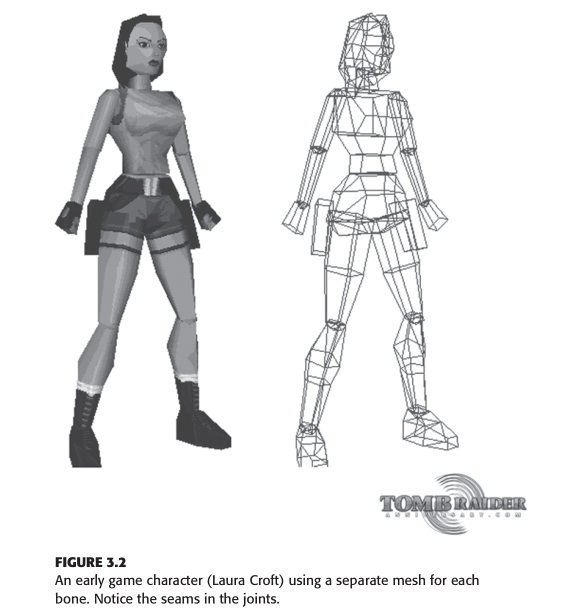
首先查看图3.1，可能最容易理解蒙皮网格的概念。在这幅画中，你可以看到一只手臂摆着三种不同的姿势，下面还有手臂对应的骨架。当肌肉围绕关节移动骨头时，“骨架”也随之移动。这个非常简单的想法是你在本章要完成的内容的精髓。



但在你看到屏幕上的任何内容之前，你需要先解决很多幕后工作。第一步是创建骨骼层次结构，方法是手动创建，或者更常见的是，通过从3D建模程序加载骨骼层次结构。之后，您需要将网格设置为骨骼层次结构。 一般来说，这可以通过软件蒙皮或硬件蒙皮以两种不同的方式完成。通过软件蒙皮，顶点的新位置由中央处理单元（CPU）在渲染角色之前计算每一帧。另一方面，通过硬件蒙皮，在渲染时间期间在图形处理单元（GPU）中即时计算顶点。我将在本章中介绍这两种技术，并查看每种技术的优缺点。 然而，首先要做的事情; 在渲染带蒙皮的角色之前，您需要创建骨骼层次结构（无论您希望采用哪种渲染方法）。

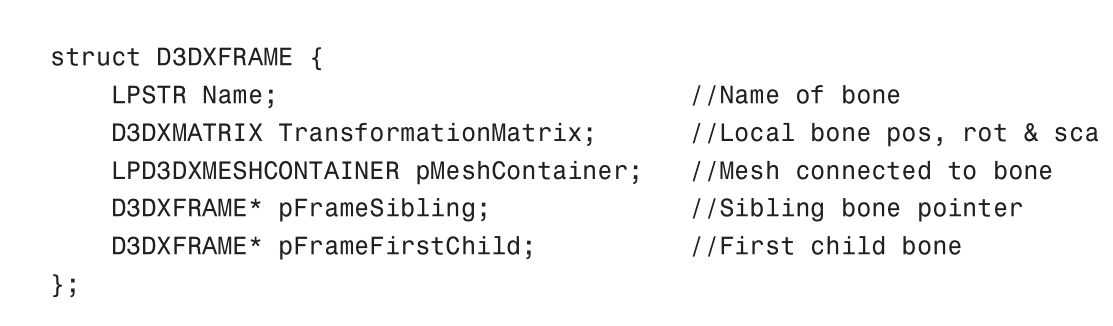
当试图理解骨骼层次结构是什么以及它是如何工作的时候，想想自己的骨骼是有帮助的。骨层次结构的基本工作原理完全相同。想想你众多的四肢之一。例如，回顾图3.1中的手臂。考虑该手臂的三个主要关节：肩部，肘部和腕关节。这三个关节用两根骨头连接起来：上臂和下臂。在骨骼层次结构中，上臂是下臂的父级。这意味着第3章蒙皮网格35图3.1人体手臂及其相应的骨骼。每当上臂围绕肩关节旋转时，这种变形也会影响下臂和手的位置（即使它们没有与父母相关地旋转）。只需移动自己的手臂就可以轻松测试自己。

如果你还记得第一次古墓丽影游戏时代的一些旧游戏，那么每个骨骼都有一个网格物体，接缝处清晰显示，如图3.2所示。



回到Laura Croft迈出第一步的那些日子里，皮肤网格对于当时的硬件而言有点过于沉重（计算方面）。但是，从早期开始，骨骼层次结构的想法并没有太大变化。然而，在本章中，目标是创建一个角色，即使在关节区域，其网格也能无缝融合。所以是时候开始构建骨骼层次结构了。但首先，需要涵盖构成层次结构中每个单独骨骼的骨骼结构。

在Direct3D中，使用D3DXFRAME结构描述了骨骼（以及其他类似的链接层次结构），并且在此结构中将构建完整的骨骼层次结构。



D3DXFRAME结构包含一个名称，正如您稍后将看到的，它可以帮助我们找到层次结构中的特定骨骼，例如头骨，手骨等。如果您想要附加不同的头盔，武器等等到角色的头部或手部，这可能很有用。

每个骨骼还有一个变换矩阵，描述其相对于其父骨骼的位置，方向和比例。 该矩阵仅描述局部空间中的变换。这基本上意味着所有位置，旋转和比例变换都围绕原点（坐标0,0,0），而不是在骨头结束的位置。如果仅使用存储在TransformationMatrix中的转换矩阵渲染蒙皮角色，那么世界的原点位置将会变得非常混乱。稍后我将扩展D3DXFRAME结构以包含完成的世界变换矩阵，我将介绍如何计算它。

再看一下D3DXFRAME结构，你会发现它还包含一个LPD3DXMESHCONTAINER指针，该指针又可以包含一个网格物体。与早期劳拉·克劳馥（Laura Croft）这样的人物一样，每个骨头都会附着一个自己的网状物。D3DXFRAME结构的工作方式类似于链接列表，其中兄弟骨骼都共享同一个父级。为了更好地理解如何使用D3DXFRAME结构构建骨骼层次结构，请参见图3.3。

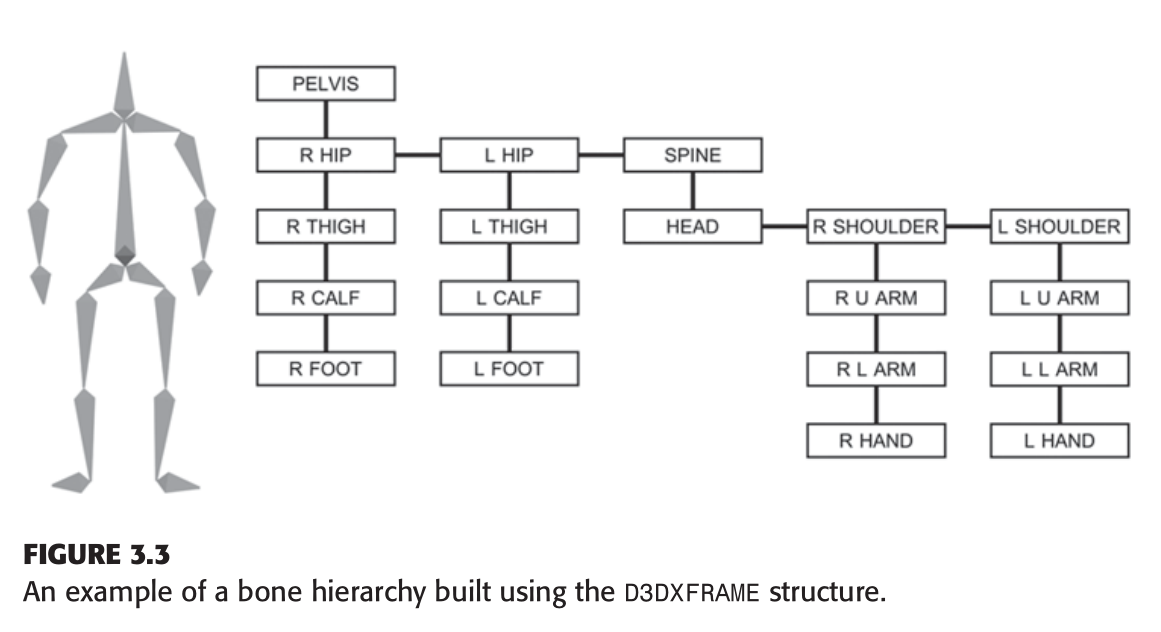
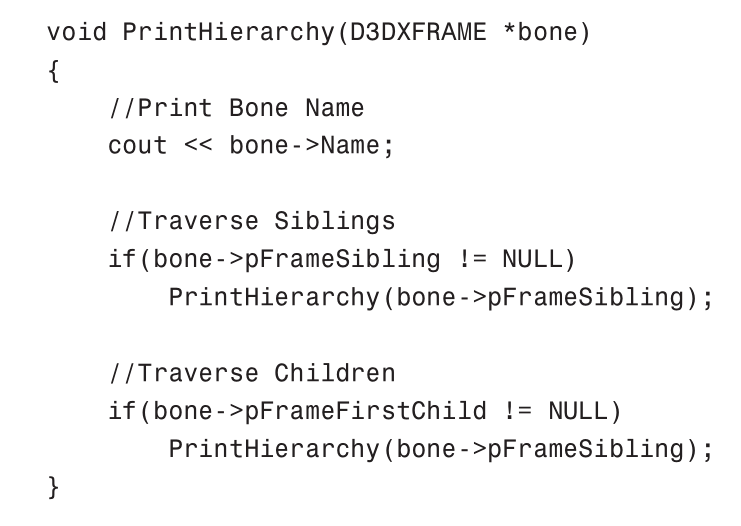
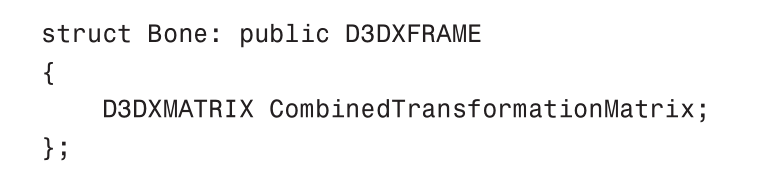


图3.3中垂直连接“第一个子”指针，水平连接“第一个兄弟”指针。如您所见，通过在每个骨骼中仅使用这两个指针，您可以描述非常复杂的骨骼层次结构。图3.3中的顶部骨骼（骨盆）是层次结构的根节点。每当遍历这样的层次结构时，始终从根节点开始。 以下代码显示如何遍历整个层次结构，遍历每个骨骼/节点。

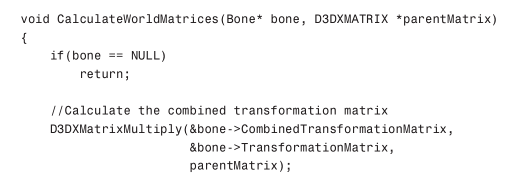


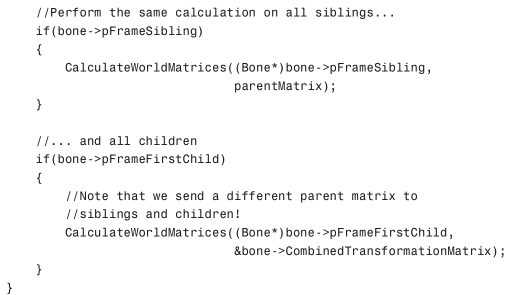
PrintHierarchy（）函数将指向D3DXFRAME对象的指针作为输入，打印其名称，然后为D3DXFRAME对象可能具有的任何兄弟和子节点调用相同的函数。希望您现在已经对如何构建和遍历复杂的骨骼结构有了一定的了解，只需使用每个骨骼的子指针和兄弟指针。

如前所述，D3DXFRAME结构只有一个变换矩阵，其中包含某个骨骼的位置，方向和比例信息。骨骼的变换矩阵与其父节点相关。在大多数情况下，骨骼的实际变换矩阵是您需要的（即骨骼的世界矩阵）。世界矩阵是将骨骼渲染到世界上正确位置所需的变换矩阵。因此，将D3DXFRAME结构进行扩展，创建如下Bone结构:



如您所见，Bone结构继承了D3DXFRAME的所有基本组件，并仅添加了CombinedTransformationMatrix变量。该矩阵将包含特定骨骼的实际世界变换。暂时，假设使用Bone对象构建的骨骼层次结构中的所有TransformationMatrix变量都包含有效的转换矩阵。然后，以下代码遍历骨骼层次结构并计算所有这些骨骼的组合变换矩阵（即，世界矩阵）。





这个函数只会在根骨节点上调用整个角色的世界矩阵。该函数将递归地遍历整个骨层次结构，用骨的实际世界矩阵填充组合转换矩阵。在继续之前，请确保您完全了解最后一个代码段。请注意，对于兄弟姐妹，CalculateWorldMatrices（）函数调用如何发送相同的父矩阵，而对于子骨骼，新计算的组合矩阵将作为参数传递。

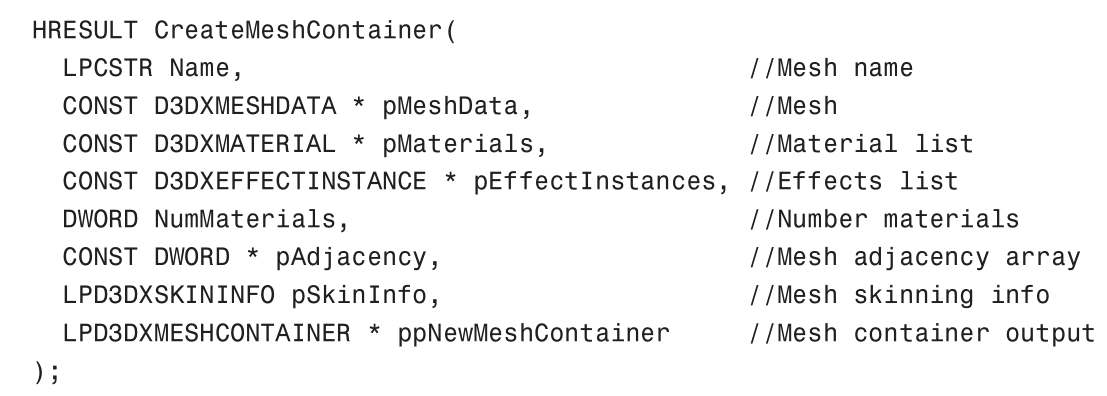
如果这是您第一次遇到链接列表和这种性质的递归函数调用，那么目前看起来有点令人困惑。我希望我能告诉你它会变得更简单。不幸的是，在角色动画方面，没有什么是那么简单。

我想我们都同意，通过代码手工构建这样的复杂层次结构将测试即使是最耐心的人的耐心。更简单，更快捷的方法是从3D建模程序导入完成的骨骼层次结构。有许多不同的文件格式可存储网格，骨骼和动画信息。在本书中，我将坚持使用.x文件格式 – DirectX自带的网格格式。有几种最常见的3D创建工具的导出器，以及针对.x文件格式的最常见文件格式之间的转换器。

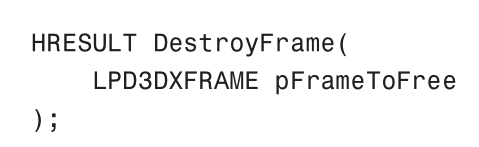
无论如何，假设您有一个皮肤完整的角色，具有骨层次结构、蒙皮信息和动画。(如果你不喜欢，随附的CD-ROM上有一些可以练习)。要加载骨骼层次结构，必须使用ID3DXAllocateHierarchy接口。这是一个非常强大的接口，并且，正如您将在本书后面看到的，这个接口不仅可以用于加载皮肤网格。这个接口的强大之处在于，您可以自己实现它的功能。虽然这需要做很多工作，但最终您将看到如何使用这个接口在加载.x文件时执行许多不同的操作。ID3DXAllocateHierarchy接口有四个您需要实现的函数：CreateFrame（），CreateMeshContainer（），DestroyFrame（）和DestroyMeshContainer（），详见以下部分。



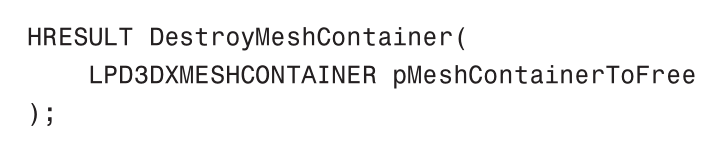
CreateFrame（）函数将指向字符串的指针和指向D3DXFRAME对象的指针作为输入。如果成功，将创建新的D3DXFRAME对象，其中变量ppNewFrame指向该对象。



如您所见，CreateMeshContainer（）函数比CreateFrame（）函数复杂得多。此方法获取名称，网格数据，材质，效果，蒙皮信息等作为输入。如果成功，此函数将返回新创建的D3DXMESHCONTAINER对象。翻回几页并查看D3DXFRAME结构，并查看此结构如何包含指向D3DXMESHCONTAINER对象的指针。 您可能已经意识到，ID3DXAllocateHierarchy从文件中读取网格和骨骼数据，并使用D3DXFRAME对象和D3DXMESHCONTAINER对象（或任何重载这些对象的用户定义结构，例如Bone结构）创建完整的层次结构。

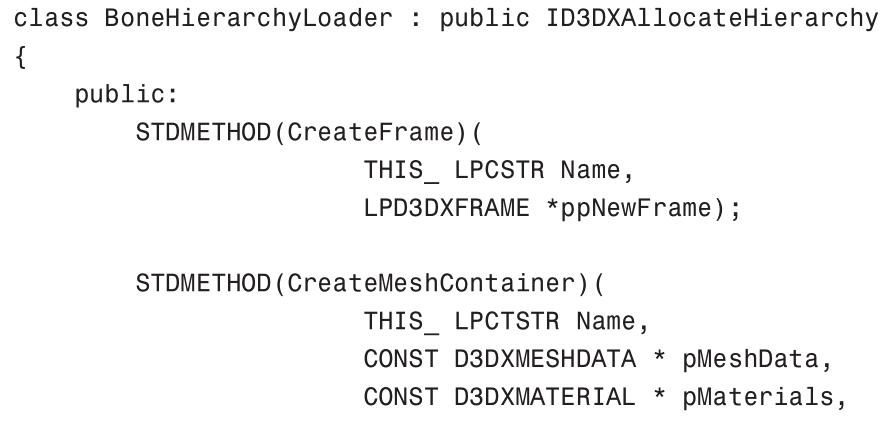


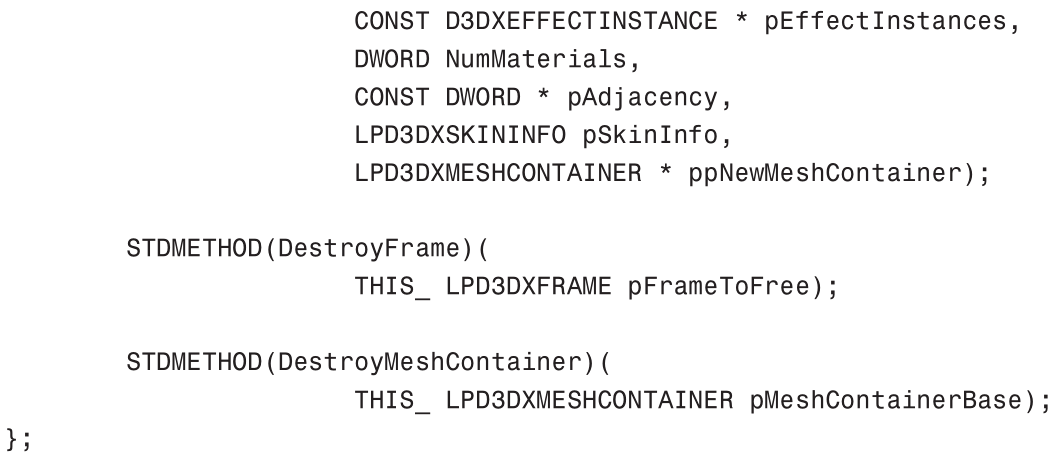
这个函数很简单。在其中，释放任何被框架(骨骼)束缚的资源。



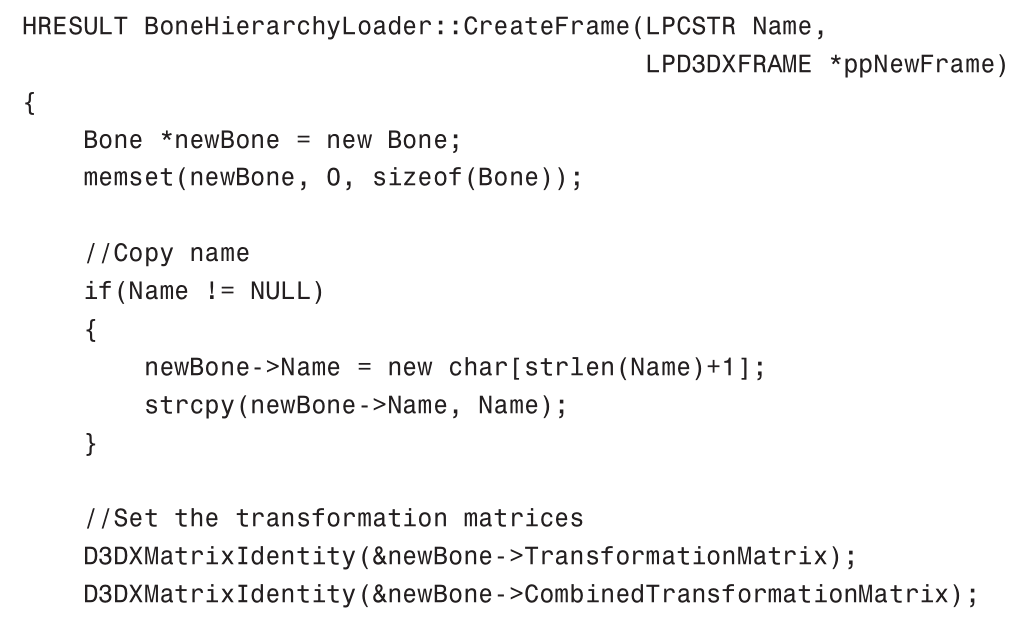
与前一个函数一样，DestroyMeshContainer（）函数旨在释放由D3DXMESHCONTAINER对象绑定的任何资源。

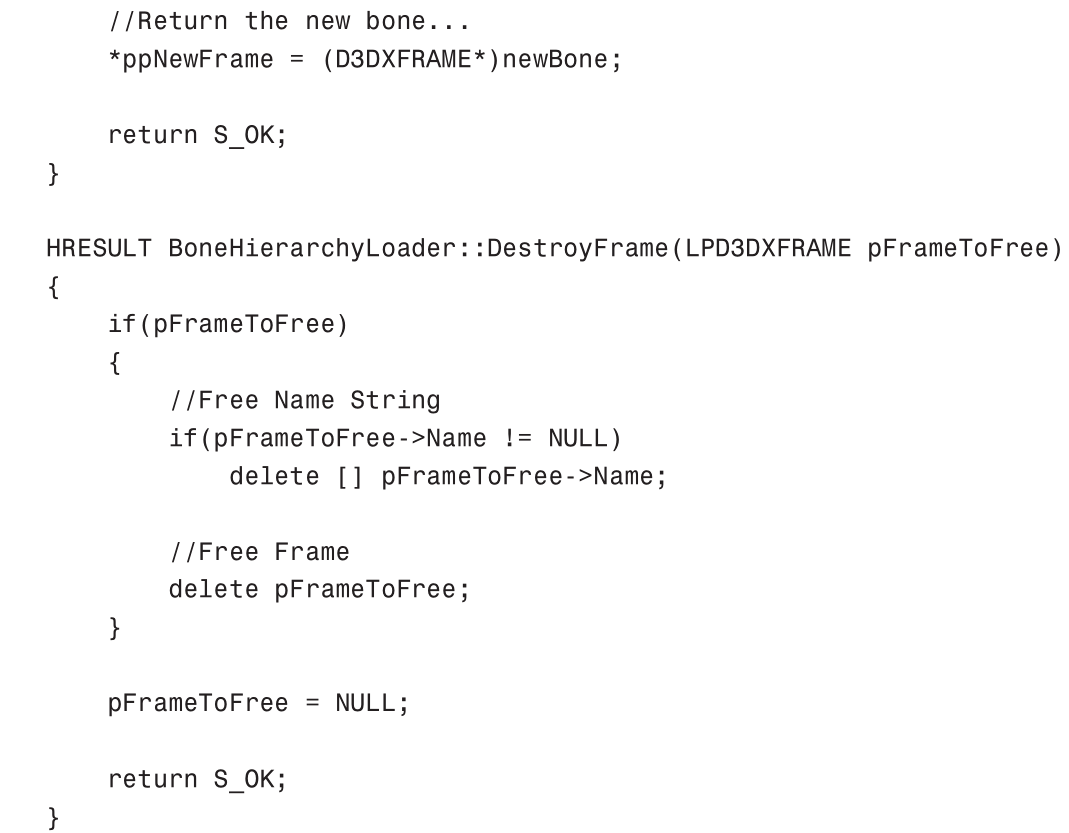
那么，为什么需要自己实现ID3DXAllocateHierarchy定义的函数呢?加载层次结构看起来非常简单。为什么它不能简单地处理这一切? 其强大之处在于，您经常想要覆盖D3DXFRAME和D3DXMESHCONTAINER结构。正如您已经看到的，骨骼结构是从D3DXFRAME结构继承而来的。通过实现ID3DXAllocateHierarchy的四个函数，您可以确切地确定创建的对象类型（以及它们的初始化方式）。要实现ID3DXAllocateHierarchy，只需创建一个继承自它的新类。 我将在本书中使用的类称为BoneHierarchyLoader，定义如下：





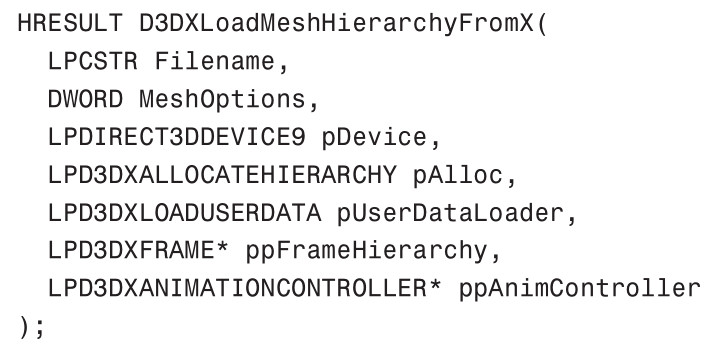
下面是CreateFrame（）和DestroyFrame（）函数的自定义实现。此实现创建了一个Bone结构而不是D3DXFRAME结构（请记住，Bone结构具有额外添加的CombinedTransformationMatrix成员来存储其世界矩阵）。





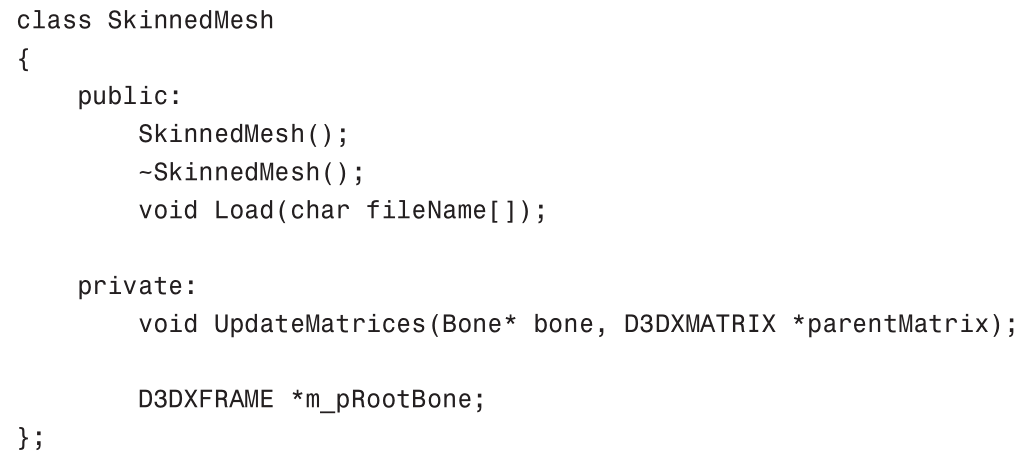
现在，我只专注于创建骨骼而不是网格容器。查看CreateFrame（）函数的自定义版本。注意如何创建Bone对象而不是D3DXFRAME对象？在此函数中，D3DXFRAME结构的成员与添加的CombinedTransformationMatrix成员一起初始化。这样，您可以创建自己的骨骼结构，并在从.x文件加载骨骼层次结构时仍使用ID3DXAllocateHierarchy接口。

查看DestroyFrame（）函数，您会看到它只释放骨骼的名称，因为这是唯一分配的内存，在删除框架本身之前不会自动释放。但是，如果您创建一个继承自D3DXFRAME结构的更高级的骨骼结构，则必须在DestroyFrame（）函数中释放在CreateFrame（）函数中分配的任何额外资源。在介绍实现ID3DXAllocateHierarchy的自定义版本的代码示例之前，这里是用于加载.x文件的D3DX函数：

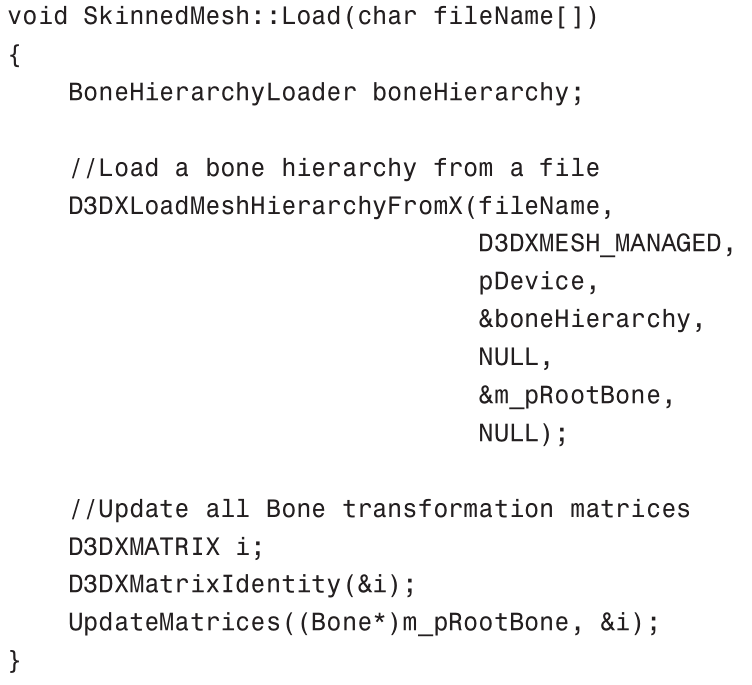


这个函数最值得注意的参数是pAlloc，ppFrameHierarchy和ppAnimController。 pAlloc是指向自定义实现的ID3DXAllocateHierarchy对象的指针（在本例中为BoneHierarchyLoader结构）。ppFrameHierarchy参数包含将存储根骨骼的位置。 ppAnimController对象基本上是用于为骨骼层次结构设置动画的结构。第4章和第5章将详细介绍ID3DXAnimationController接口，您将学习如何为角色设置动画。有关其他参数，请参阅DirectX文档。

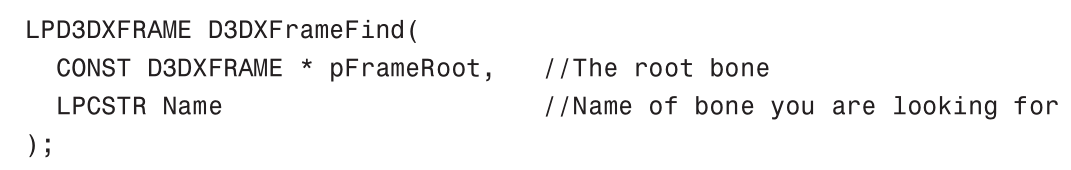
要包含和封装用于蒙皮角色的所有必要数据，将创建以下SkinnedMesh类：



正如您所看到的，这个类目前没有包含太多内容。稍后，随着更多的功能添加到蒙皮角色中，这个类将被扩展。load函数基本上封装了D3DX库函数d3dxloadmeshhierarchy yfromx()。几页前，当您学习如何计算层次结构中每个骨骼的组合转换矩阵时，UpdateMatrices（）函数已被覆盖。另请注意，目前SkinnedMesh类中唯一的成员是指向D3DXFRAME对象的指针（稍后将保留整个骨骼层次结构的根骨骼）。以下代码显示了SkinnedMesh加载函数：

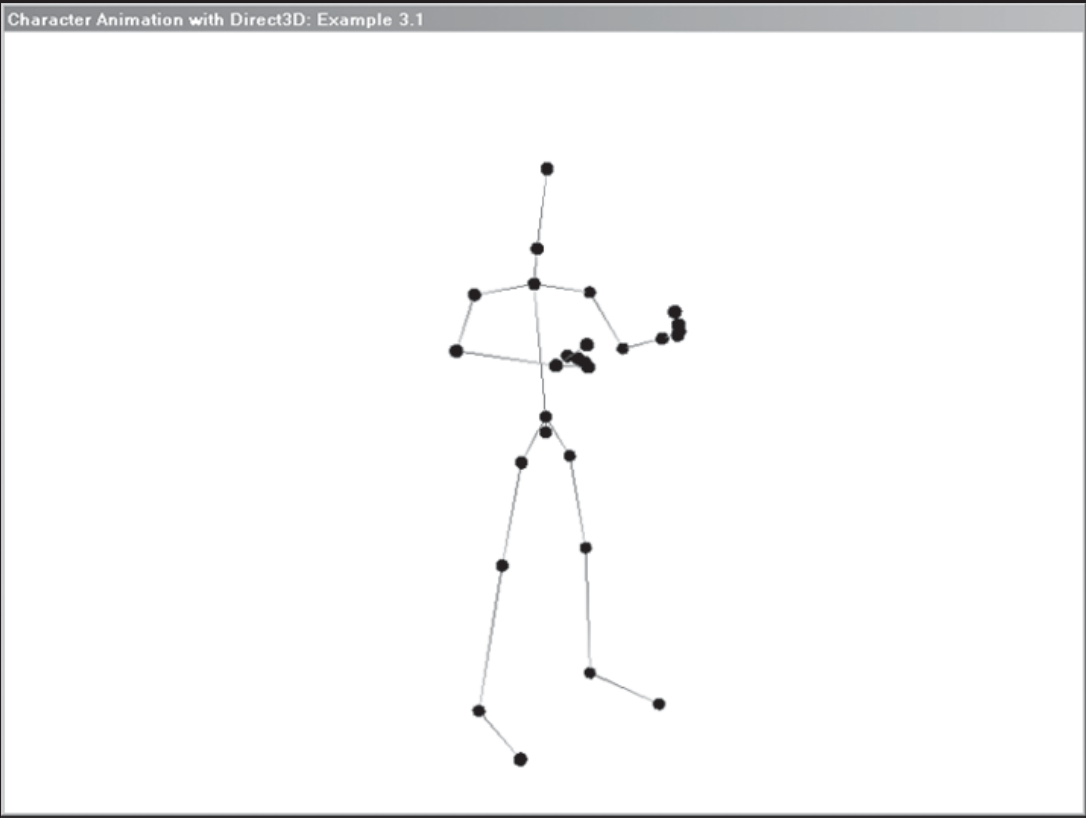


有时，在层次结构中定位特定骨骼会很有用 - 例如，如果您想要找到角色的颈骨并应用旋转变换矩阵并使头部转动。以下D3DX功能非常有用：

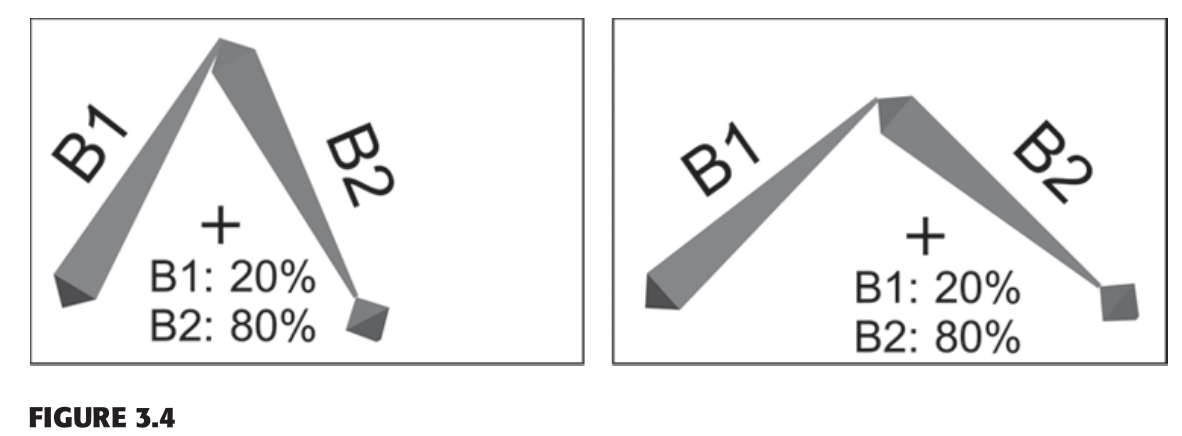


此函数返回指向层次结构中正确骨骼的指针，如果未找到骨骼，则返回NULL。尝试在Example 3.1中使用此函数来查找颈骨。

希望您现在已经知道如何通过实现ID3DXAllocateHierarchy接口来加载骨骼层次结构。在本书的后面，您将看到如何使用相同的界面从单个.x文件加载多个不同的变形目标，而不是将这些网格保存在单独的文件中。但是，现在是时候将网格实际应用于骨骼层次结构了。



您可能知道，网格由多个多边形组成，而这些多边形又由一个或多个三角形组成。每个三角形依次由三个顶点定义，即三维空间中的三个点。在研究如何将复杂的角色网格皮肤化为骨骼层次之前，先看看单个顶点。一个顶点可以被骨骼层次结构中的一个或多个骨骼链接(影响)。骨骼影响顶点的数量由权重值决定，如图3.4所示。

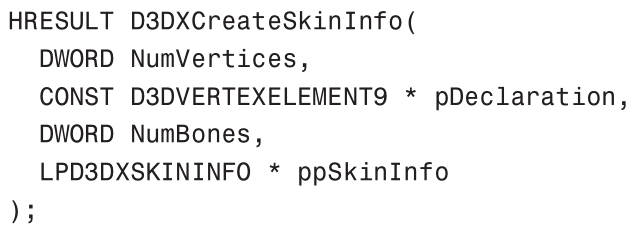


顶点（十字）如何受到两个骨骼（B1和B2）影响的示例，其中权重分别为20％和80％。顶点（十字）如何受到两个骨骼（B1和B2）影响的示例，其中权重分别为20％和80％。 注意由于权重，顶点如何跟B1比B1更多。

在更多数学术语中，应用于顶点的变换矩阵定义如下：



对于所有影响骨骼，此公式将骨骼重量（w x）与骨骼变换矩阵（M x）相乘，并对结果求和（M Tot）。然后使用所得到的矩阵来变换顶点。在DirectX中，关于哪些骨骼影响哪些顶点以及它们各自的权重等信息通过ID3DXSkinInfo接口存储和控制。创建此接口的一种方法是使用以下D3DX函数：



此函数获取网格中顶点的数量、顶点声明（即每个顶点包含的信息）以及用于蒙皮网格的骨骼数量。如果您在代码中制作需要蒙皮的东西，这将是最好的方法。然而，角色一般会在3D软件（如3D Studio Max、Maya或类似软件）中创建和蒙皮。幸运的是，当您将这样的角色导出为.x文件格式时，同时还会导出皮肤信息。如果再次查看ID3DXAllocateHierarchy接口的CreateMeshContainer（）函数，您会注意到此函数的其中一个参数确实是指向ID3DXSkinInfo对象的指针。因此，在读取.x文件时，您需要做的就是存储此对象，并在之后对角色进行蒙皮时使用它。

接下来就会实现CreateMeshContainer（）函数，通过它可以完成加载带有骨骼的角色的过程。但是，首先，您需要了解如何渲染蒙皮角色的两个主要选择。第一个选项是使用CPU进行蒙皮又名软件蒙皮。另一个选择是在渲染网格时直接用GPU（图形处理单元，即图形卡）进行蒙皮又名硬件蒙皮。（这两种技术还有其他变体，但这两种是主要选择）。

使用第一个选项软件蒙皮，使用前面介绍的数学公式计算网格中每个顶点的位置。结果存储在临时网格中，然后进行渲染。与硬件蒙皮相比，简单，直接，但也非常慢。因此，如果它与硬件蒙皮相比这么慢，为什么要使用它呢？事实上，角色作为网格存储在内存中是软件蒙皮的主要优点。使用这个临时网格物体，阴影投射，拾取等等变得更容易一些。通过软件蒙皮，影响顶点的骨骼数量也无关紧要。如果您正在制作第一人称射击游戏（FPS）游戏，你可能想要测试你所发射的子弹是否击中了敌方士兵。通过软件蒙皮，使用简单的网格射线交叉测试就可以很容易的进行测试。

使用硬件蒙皮，当然也可以进行阴影投射，拾取等，但是需要更多功夫才能使其工作。硬件蒙皮还有一些限制，即有多少骨骼可以影响顶点，以及每个角色可以拥有多少骨骼，而不必将网格分成几个部分。虽然硬件蒙皮功能上损失的一些但是其在速度上是要有优势的。所以请记住根据您正在制作的特定游戏选择您的蒙皮方法。在以下两节中，硬件蒙皮和软件蒙皮将会有更多的细节展示。

让我们首先看看软件蒙皮，因为这是一种更直接，更简单的实现方法。以下是使用软件蒙皮渲染蒙皮网格所需步骤的简要概述：

1.（可选）重载D3DXFRAME

2.（可选）重载D3DXMESHCONTAINER

3.实现ID3DXAllocateHierarchy接口

4.使用D3DXLoadMeshHierarchyFromX（）函数加载骨骼层次结构和关联的网格，蒙皮信息等

5.对于每个帧，更新骨架姿势（即SkinnedMesh :: UpdateMatrices（）函数）

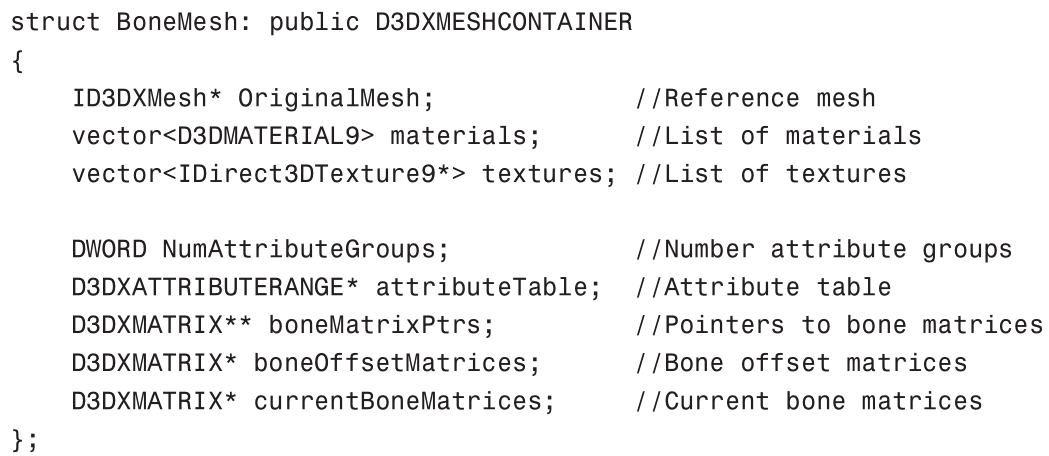
6.使用ID3DXSkinInfo :: UpdateSkinnedMesh（）更新目标网格

7.将目标网格渲染为常见静态网格

蒙皮网格路径的第一步是创建自己的网格容器结构。您可以通过重载D3DXMESHCONTAINER结构来执行此操作，该结构定义如下：

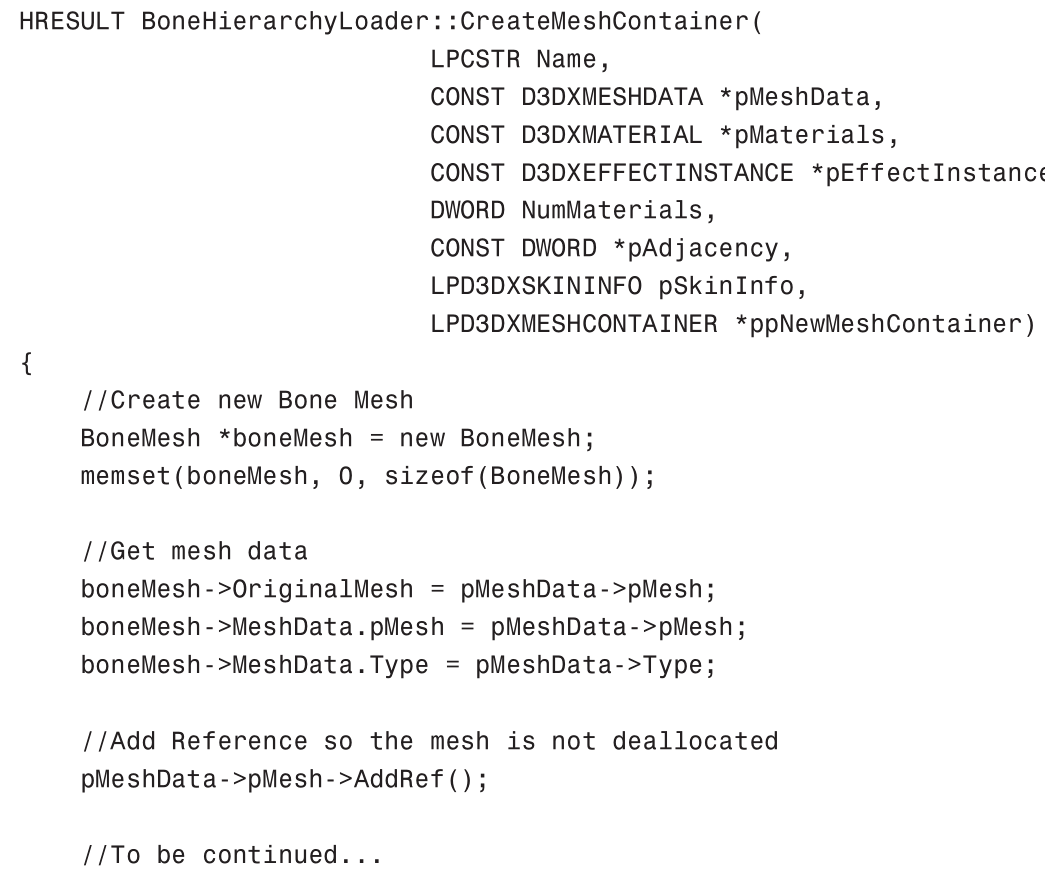


D3DXMESHCONTAINER包含网格本身（在D3DXMESHDATA结构中）以及渲染网格（材质，纹理和着色器）所需的所有必要内容。纹理文件名作为D3DXMATERIAL结构的成员存储，必须单独加载。此结构的另一个值得注意的成员是pSkinInfo变量，它将包含已加载的任何蒙皮网格的蒙皮信息。但是，我们希望在此结构中添加一些内容，以便使用软件蒙皮更容易渲染网格。因此我创建了BoneMesh结构，定义如下：

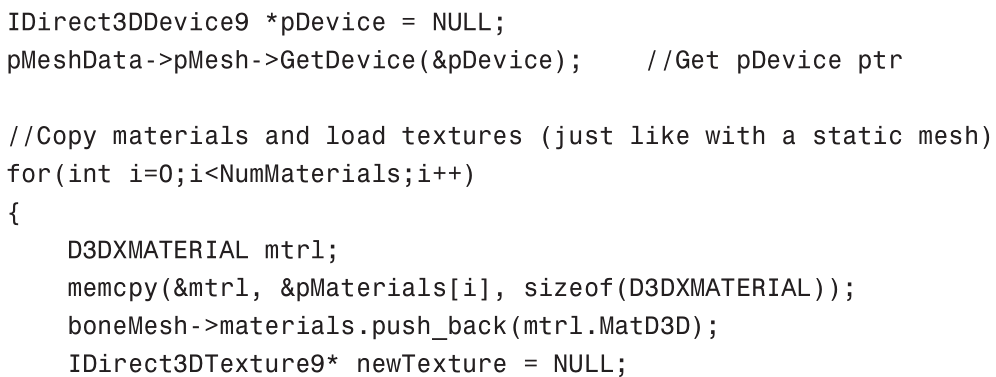


如您所见，与Bone结构中添加的内容相比，BoneMesh结构中添加了大量额外信息。前三个成员应该很容易理解; 其他成员变量可能看起来有点晦涩。表3.1提供了有关成员的更多详细信息。（表没有。。。。。。。。）

现在您已经看到了重载的D3DXMESHCONTAINER结构，看看如何实现ID3DXAllocateHierarchy的CreateMeshContainer（）函数以将网格加载到BoneMesh对象中。

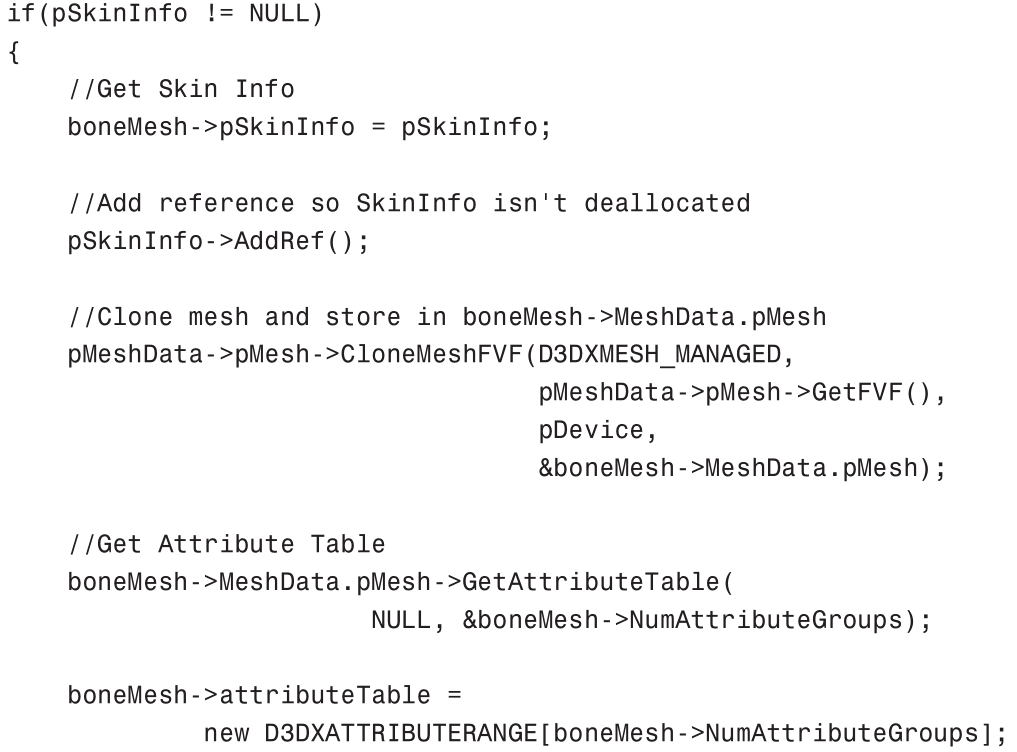


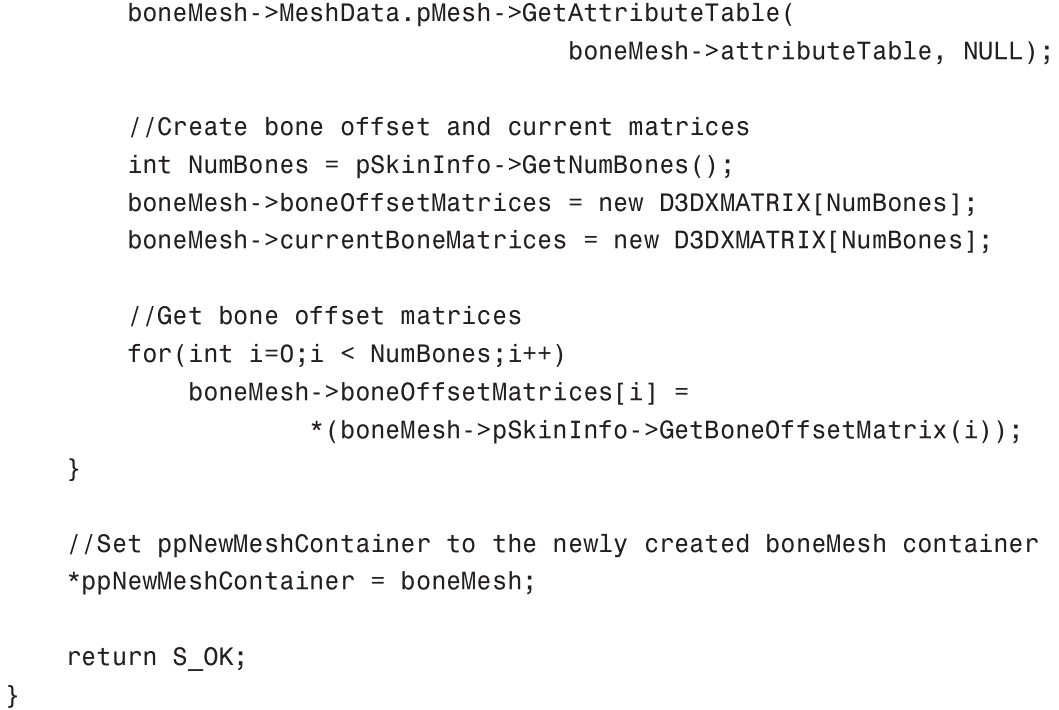
首先，使用memset（）函数创建一个新的BoneMesh对象，并将其所有成员设置为零和NULL。接下来，将引用添加到OriginalMesh和MeshData成员的网格数据中。在进行软件蒙皮时，需要保持网格的原始形式的副本（当涉及到网格的渲染时，将会说的更加详细）。





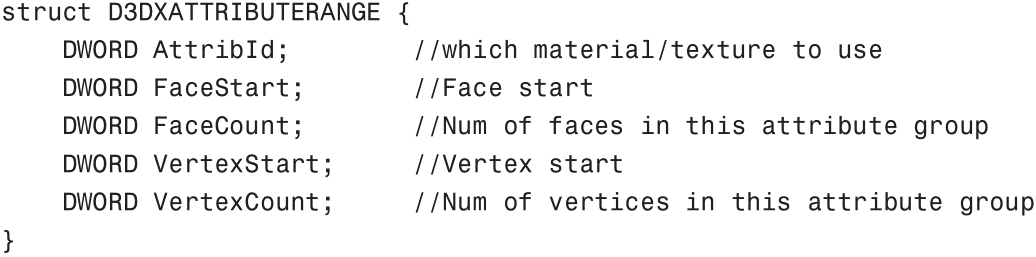
在CreateMeshContainer（）函数的这一部分中，您首先获得指向当前设备的指针。之后，将所有材质复制到BoneMesh结构，如有必要，任何需要的纹理都可以与相关的材质一起加载（这就是您需要检索设备指针的原因）。接下来，存储作为参数发送到CreateMeshContainer（）的皮肤信息。





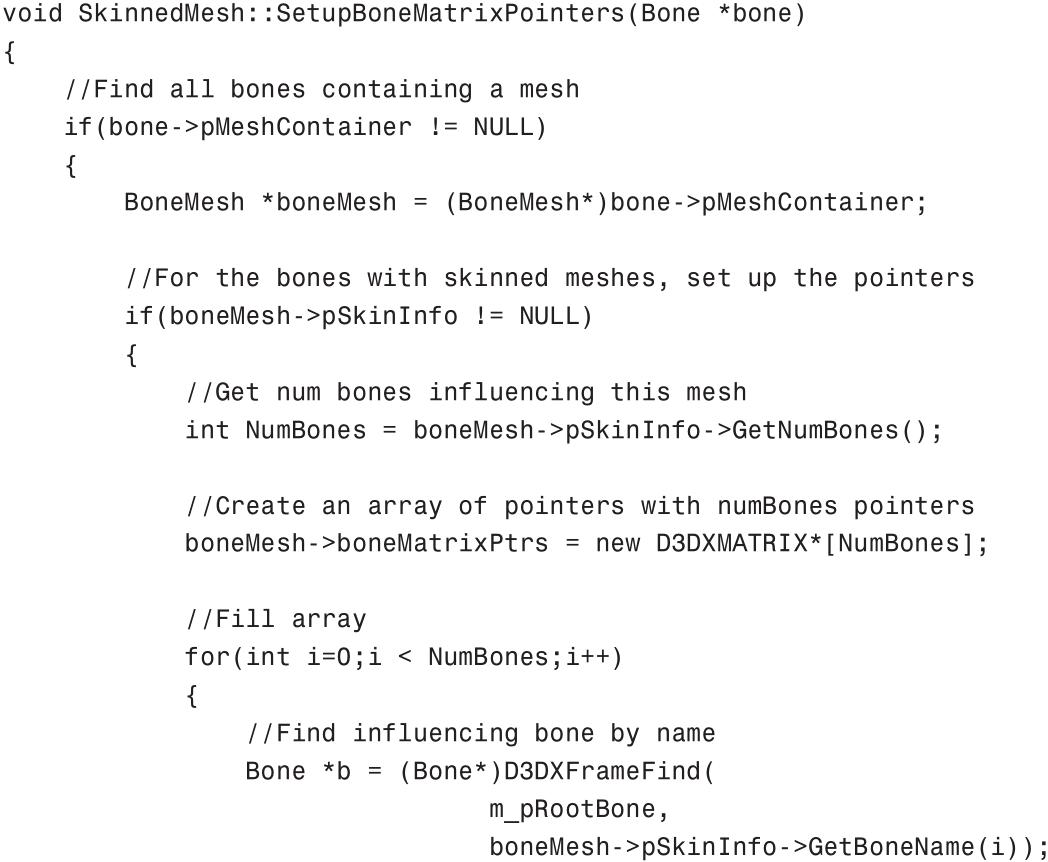
在CreateMeshContainer()函数的最后一部分中，检查是否有可用的皮肤信息。如果有，将其复制存储到BoneMesh结构的pMeshData成员中的网格里。克隆的网格稍后将是渲染的实际蒙皮网格。还要记住存储指向原始网格（OriginalMesh成员）的指针。这个网格将作为一个引用来创建存储在pMeshData中每个帧的蒙皮网格。在这段代码中，存储了属性组的数量以及属性表本身。 然后根据蒙皮信息中定义的骨骼数量创建矩阵数组（注意，您要从ID3DXSkinInfo对象复制骨骼偏移矩阵）。最后，您只需存储创建的BoneMesh对象并返回S\_OK。

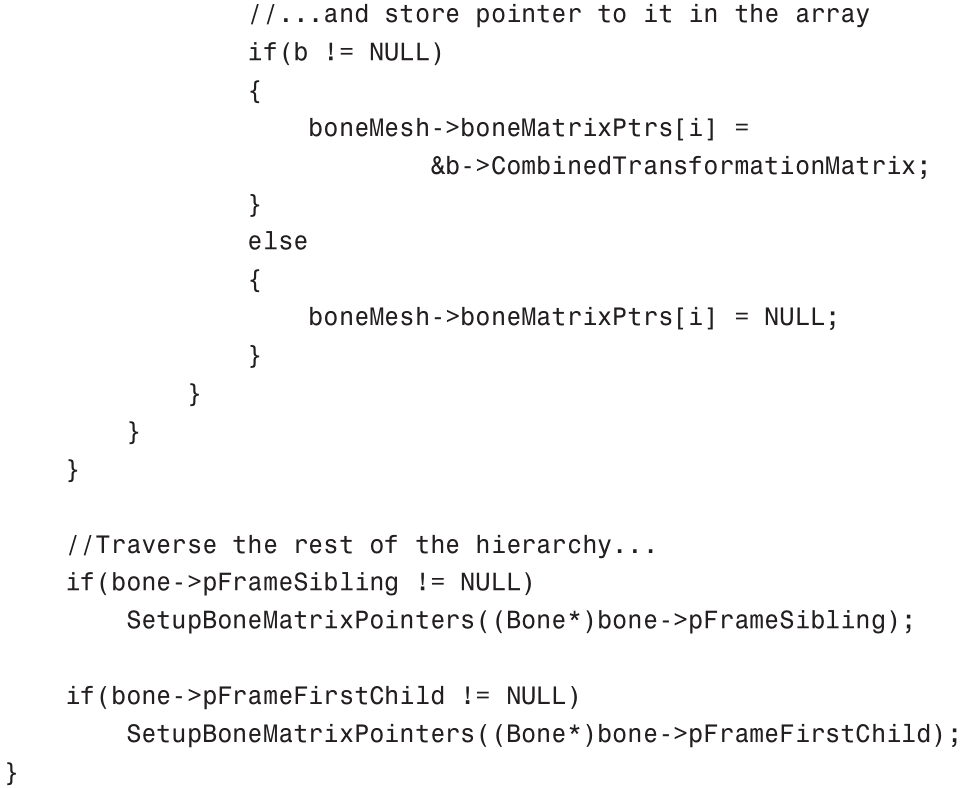
以下是使用D3DXATTRIBUTERANGE对象数组存储属性表：



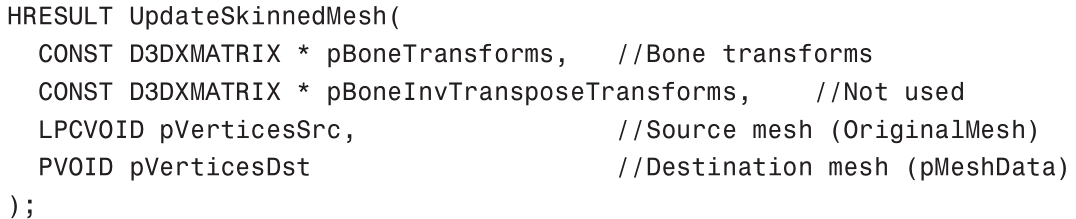
稍后，当您渲染网格时，循环遍历属性表，获取属性，并以此设置要用于渲染网格子集的材质和纹理。一般渲染角色时可以有多种材质和纹理组合。

现在您知道如何使用扩展的Bone和BoneMesh结构以及BoneHierarchy-Loader加载骨骼层次结构和附加到骨骼的任何网格。现在你终于到了屏幕上可以看到蒙皮角色的地步了。要渲染BoneMesh，我们需要计算所有影响骨骼的当前矩阵，并将它们存储在BoneMesh boneMatrixPtrs数组中。因此，在使用D3DXLoadMeshHierarchyFromX（）函数加载网格后，我们调用以下函数来设置这些矩阵指针：



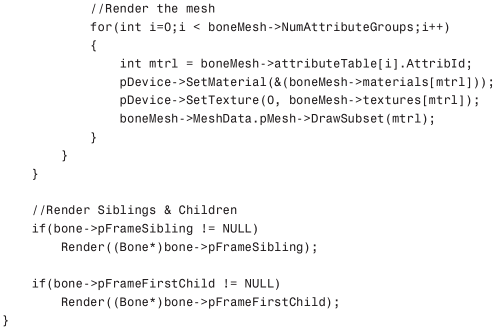


此函数查找影响某个BoneMesh的所有骨骼，并存储指向其CombinedTransformationMatrix（即世界矩阵）的指针。因此，在更新骨架并放入某个姿势后，可以在渲染角色期间通过此数组访问这些世界矩阵。然后，要在软件中更新网格，您需要使用ID3DXSkinInfo :: UpdateSkinnedMesh（）函数：

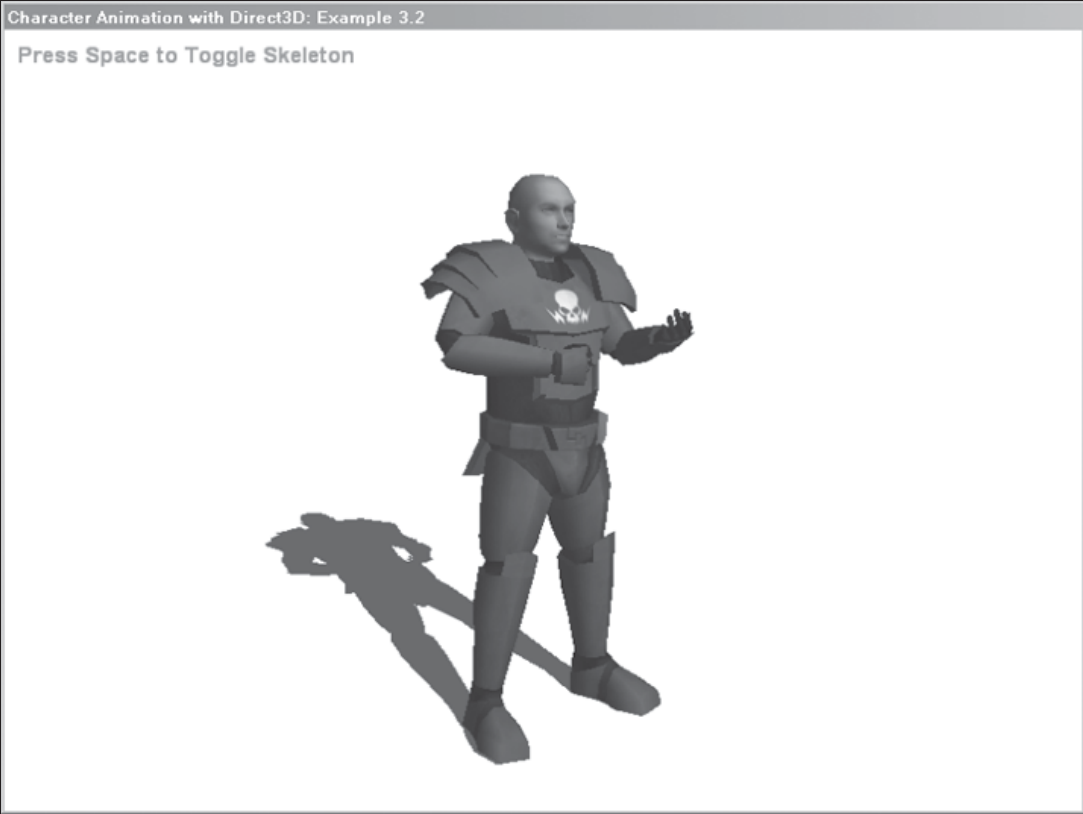


pBoneInvTransposeTransforms参数在这里看起来有点奇怪。但是，仅当您具有包含两个位置元素的顶点时才使用此选项。在那种情况下，这组变换矩阵用在第二位置元素上。但是，在本书中，您不需要使用它，因此只需将此参数设置为NULL即可。我希望你能记住用于封装骨骼层次结构和加载函数的SkinnedMesh类。现在，Render（）函数将添加到此类中。在此函数中，将使用在D3DXSkinInfo接口中定义的UpdateSkinnedMesh（）函数更新蒙皮网格。然后网格将被如下渲染：





SkinnedMesh :: Render（）函数将Bone指针作为参数。如果此Bone包含BoneMesh，则渲染它，然后在骨骼可能具有的任何子项或兄弟上调用Render（）函数。通过这种方式，您可以遍历整个骨骼层次结构，从而渲染沿途找到的任何BoneMesh对象。首先设置BoneMesh的当前矩阵，然后锁定源网格（OriginalMesh）和目标网格（pMeshData）的顶点缓冲区。在此之后，调用UpdateSkinnedMesh（）函数，该函数根据骨架的当前姿势计算网格中每个顶点的新位置。更新网格后，使用在加载网格期间存储的属性表渲染其每个子集。



一旦你掌握了（或者至少在某种程度上理解了）软件蒙皮的过程，就该看看它不那么温柔的表兄：硬件蒙皮了。通过硬件蒙皮，网格在GPU中即时进行蒙皮，而不是像软件蒙皮那样对每个帧进行预处理。虽然实现比软件蒙皮更棘手，但硬件蒙皮处理速度要快得多。两种蒙皮方式中虽然有些事情是不同的，但主要的区别是硬件蒙皮使用顶点着色器进行蒙皮计算。在您查看提供着色器所需的信息之前，先看看硬件蒙皮所需的步骤。

1.（可选）重载D3DXFRAME

2.（可选）重载D3DXMESHCONTAINER

3.实现ID3DXAllocateHierarchy接口

4.使用D3DXLoadMeshHierarchyFromX（）函数加载骨骼层次结构和关联的网格，蒙皮信息等

5.将网格转换为Index Blended Mesh

6.对于每个帧，更新骨架姿势（即SkinnedMesh :: UpdateMatrices（）函数）

7.将Matrix Palette（列如骨骼矩阵）上载到顶点着色器

8.使用顶点着色器渲染Index Blended Mesh

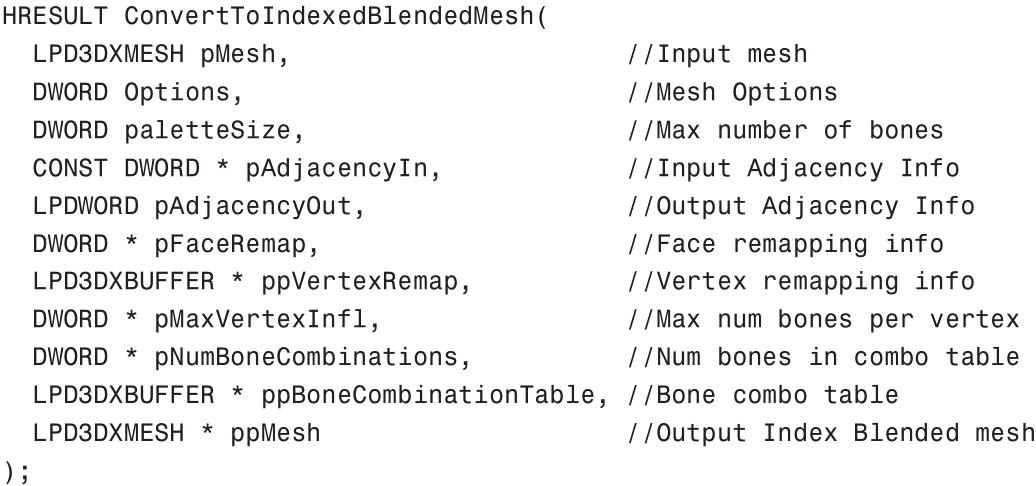
如您所见，此处的大多数步骤与软件蒙皮相同。这里有两个新概念：Matrix Pale（矩阵调色板）和Index Blended Mesh（索引混合网格）。在查看创建这些代码的代码之前，先来快速浏览一下它们背后的理论。

在软件蒙皮中，当前骨骼转换矩阵的数组被用作ID3DXSkinInfo :: UpdateSkinnedMesh（）函数的参数。此函数使用存储在ID3DXSkinInfo对象中的信息来匹配所有顶点与影响它们的骨骼以及它们相应的权重。现在你必须在顶点着色器中自己处理这个计算。Matrix Palette只是当前骨骼转换数组的另一个名称。幸运的是，这可以在将网格转换为索引混合网格的同一步骤中完成。

既然您原则上知道如何构建Matrix Palette以及顶点着色器常量带来了哪些限制，那么您可以将网格转换为Index Blended Mesh（后文皆为索引混合网格）。

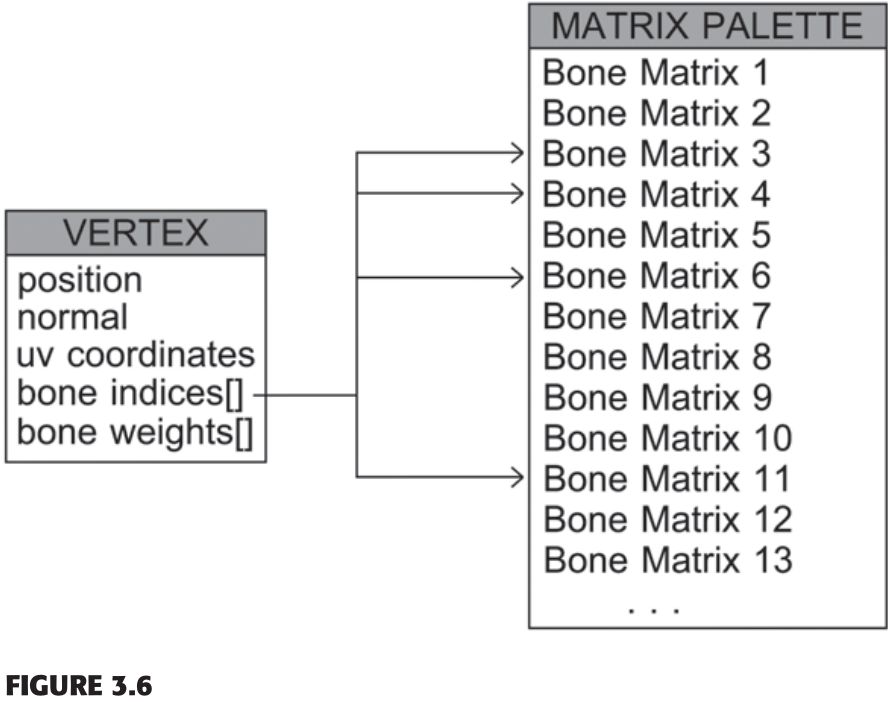
图3.5显示了ID3DXSkinInfo :: ConvertToIndexedBlend dMesh（）函数如何将顶点转换为Index Blended Vertex。它将骨骼权重和骨骼索引添加为顶点元素。它还将这些索引设置为指向Matrix Palette中正确的骨骼转换矩阵，如图3.6所示。

在这种情况下，使用四个骨骼（即，骨骼3,4,6和11）混合顶点。每个骨骼的权重作为浮点值数组存储在顶点本身中。要进行此转换，您需要做的就是在CreateMeshContainer（）函数中调用ConvertToIndexedBlendedMesh（）函数。



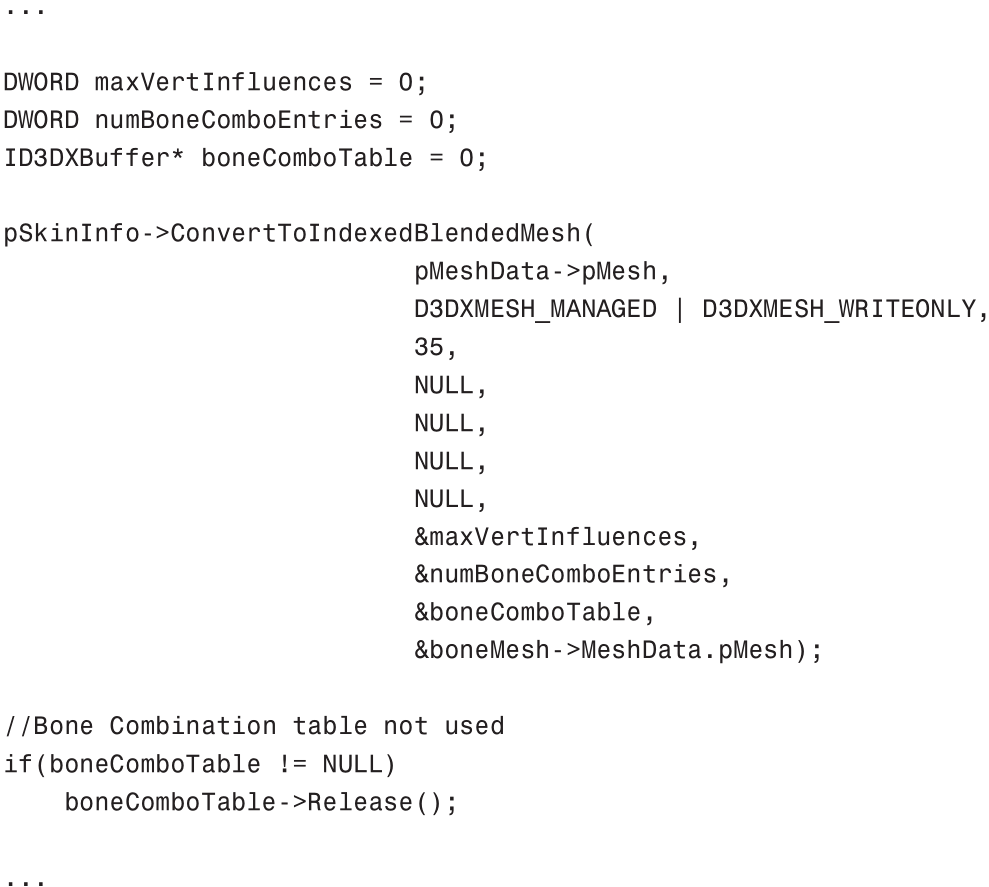


此图显示了如何转换顶点以包含硬件蒙皮所需的信息。

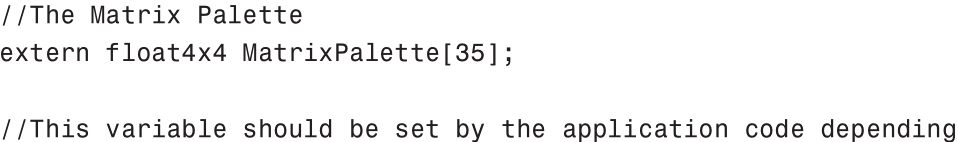


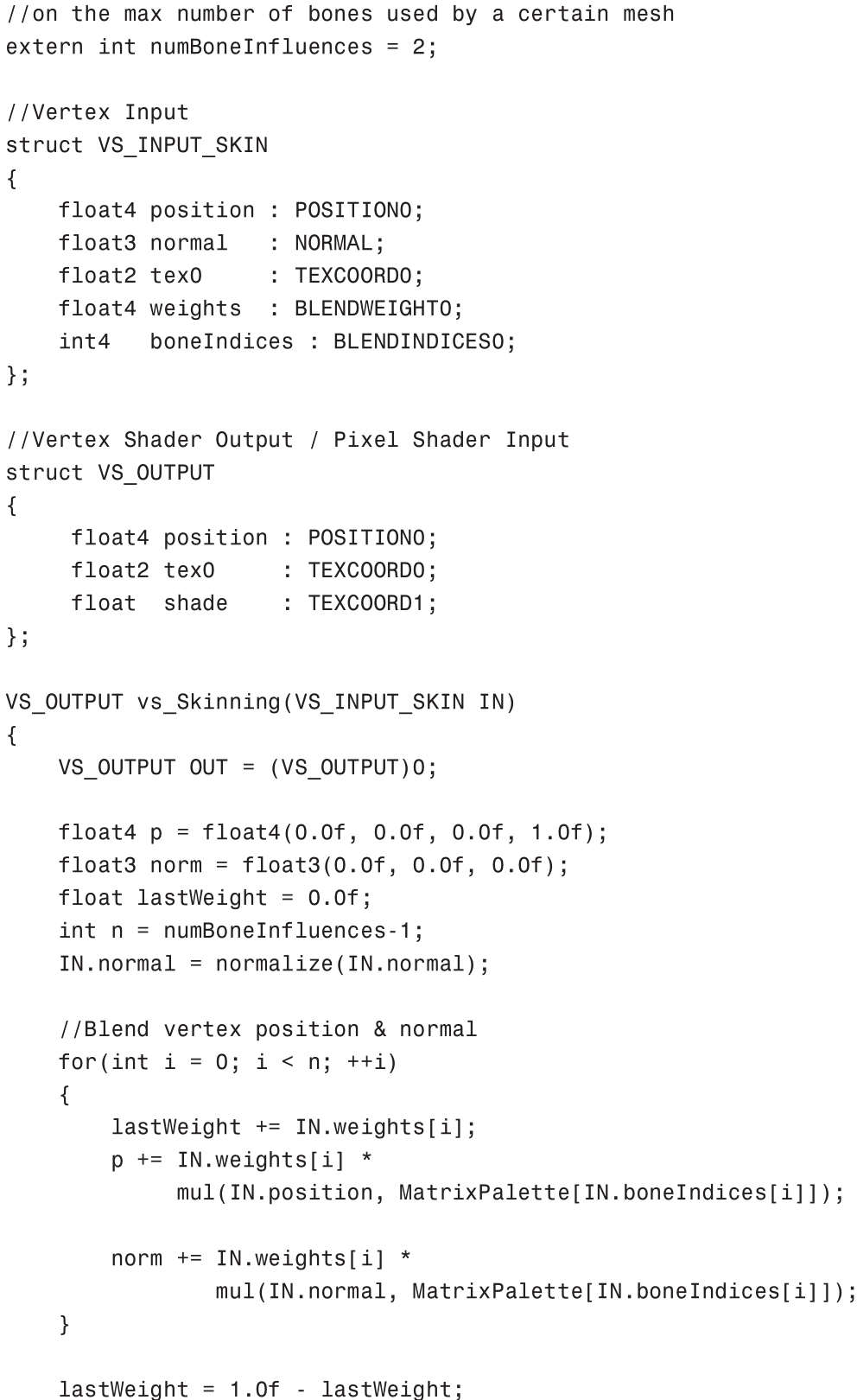
该图显示了单个顶点和Matrix Palette之间的关系。

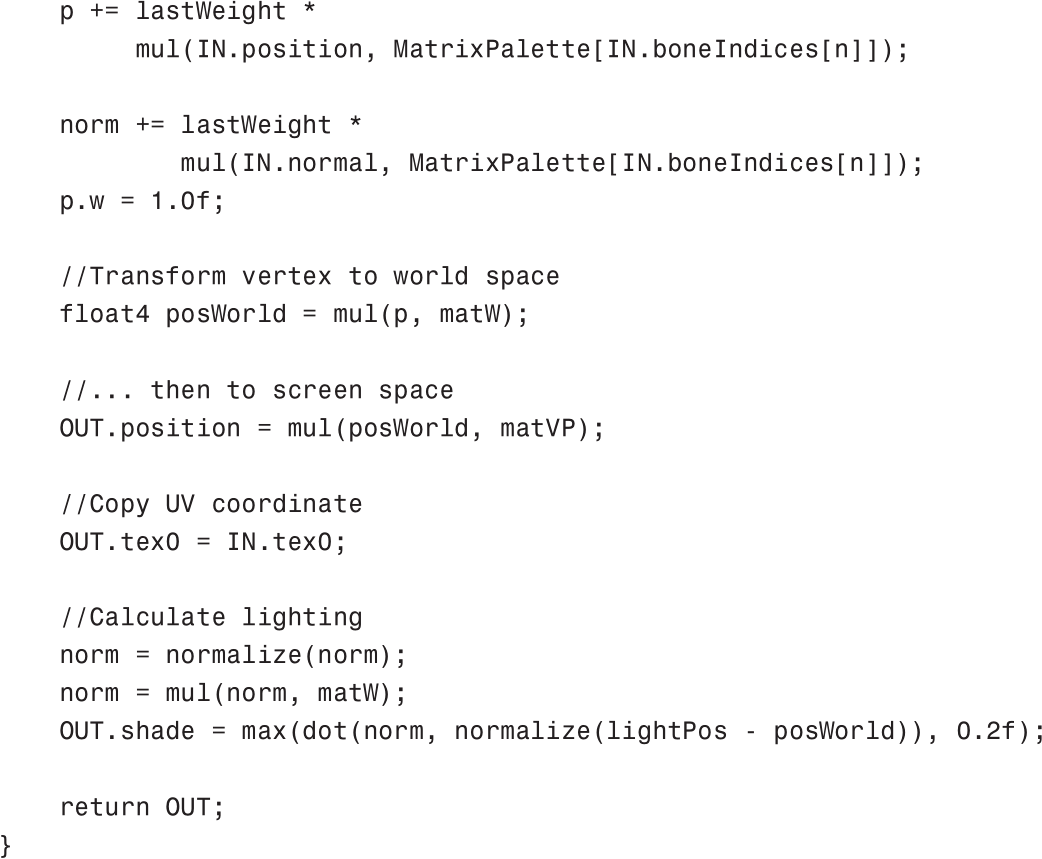
其中一些参数您无需担心。但是，您需要指定可以使用的最大骨骼数（即Matrix Palette的最大大小）。如果骨骼数量超过此值，则ConvertToIndexedBlendedMesh（）函数会将网格划分为多个部分。有关这方面的更多信息，请查看DirectX文档。以下是CreateMeshContainer（）的摘录，其中进行了必要的更改以适应硬件蒙皮：



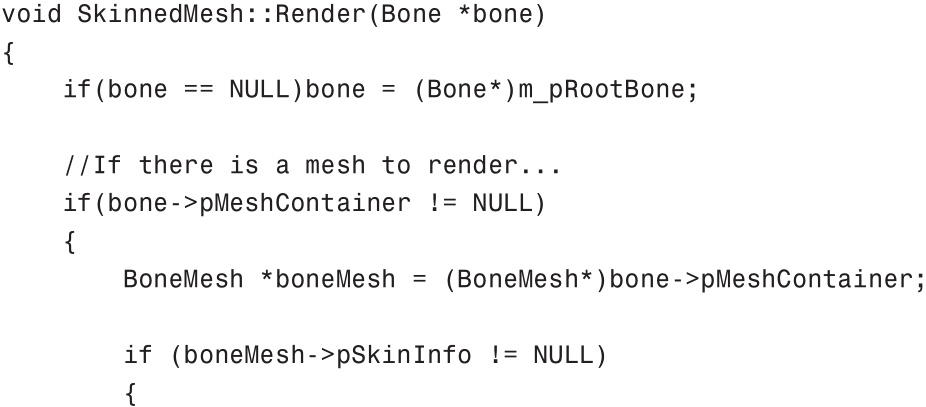
嗯，那很简单，不是吗？现在，网格已经完全转换为索引混合网格，只需一个简单的函数调用。但是，了解此函数的作用非常重要，因为现在您将直面执行蒙皮的实际顶点着色器。现在，您将了解本书中第一个正确的着色器。我假设你已经足够了解高级着色语言（HLSL）来阅读这个着色器并理解它。如果没有，有几个很好的资源，你可以开始学习它。 在此摘录仅涵盖顶点着色器。请参阅稍后介绍的完整效果代码（.fx）示例，包括像素着色器代码以及它如何合并到应用程序代码中。





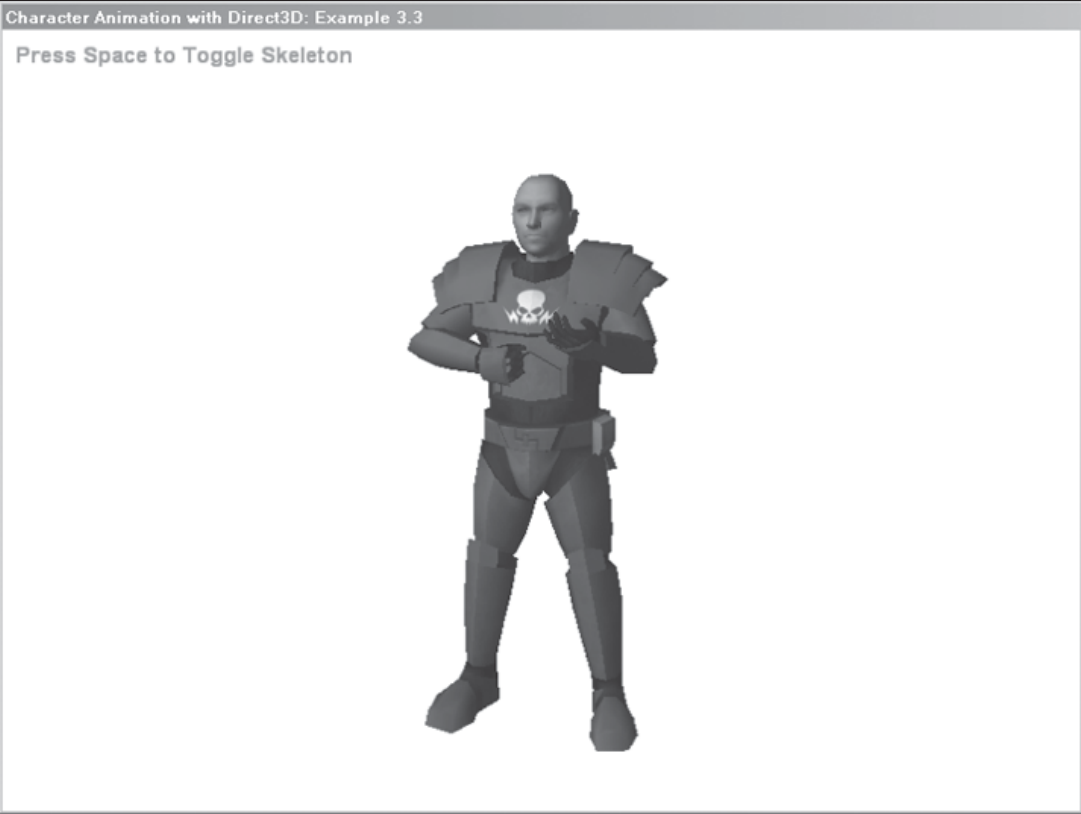


此高级着色语言（HLSL）着色器仅支持最多四个影响单个顶点的骨骼。注意顶点输入结构; 查看权重和骨骼指数以及它们在着色器中如何使用它们来索引和称量Matrix Palette中的骨骼矩阵。另请注意，您手动计算最后一个重量。这是为了确保权重总和总是达到1.0f。一旦着色器到位，唯一剩下的就是编辑SkinnedMesh :: Render（）函数，以便它使用顶点着色器进行蒙皮：

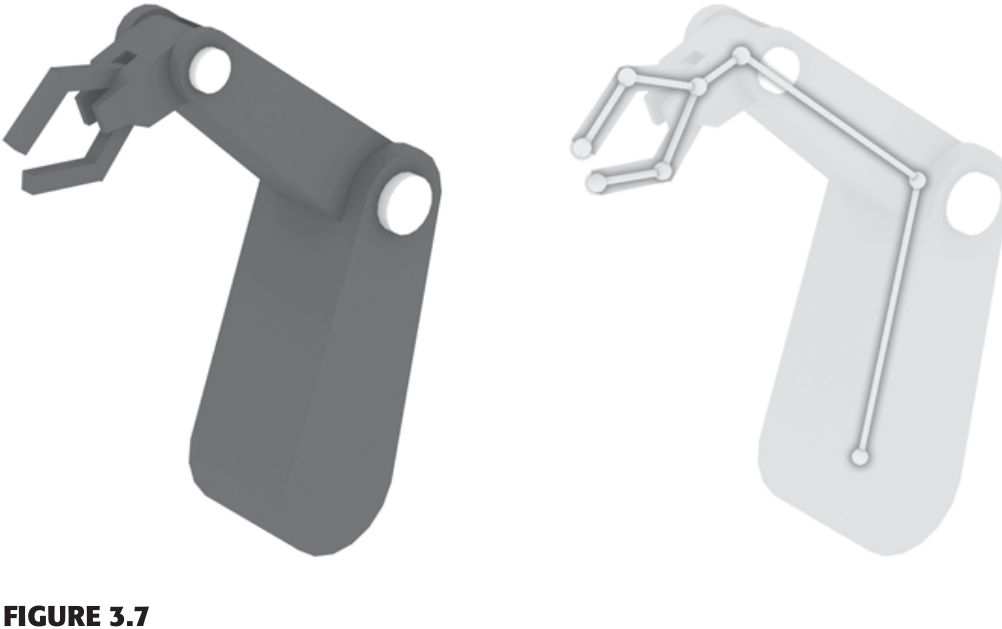




与软件皮肤示例相比，此功能没有太大变化。 当然，最值得注意的是使用着色器并将Matrix Palette上传到着色器。另外，循环遍历网格的不同属性组并使用着色器渲染它。



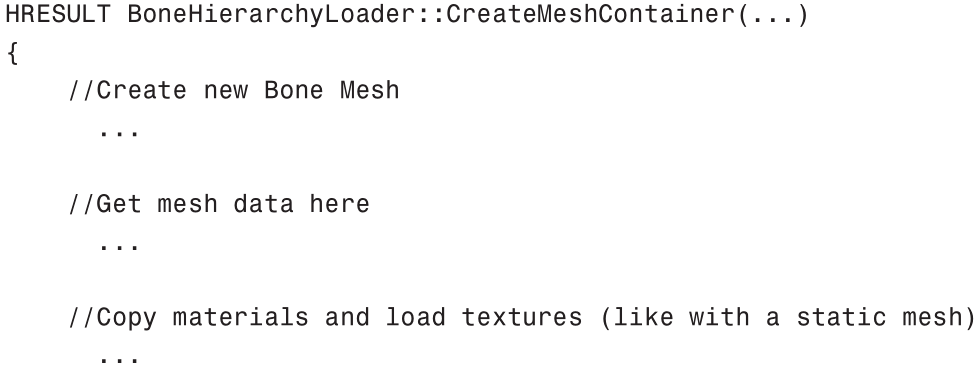
有时您可能不希望角色被蒙皮。制作机器动画就是一个很好的例子。机械很少有“软”部件; 因此，你并不需要在你的游戏里用一个蒙皮网格来制作机械。尽管如此，您可能希望有一个骨骼层次结构来控制“机器”的不同部分。以机器人手臂为例，如图3.7所示。

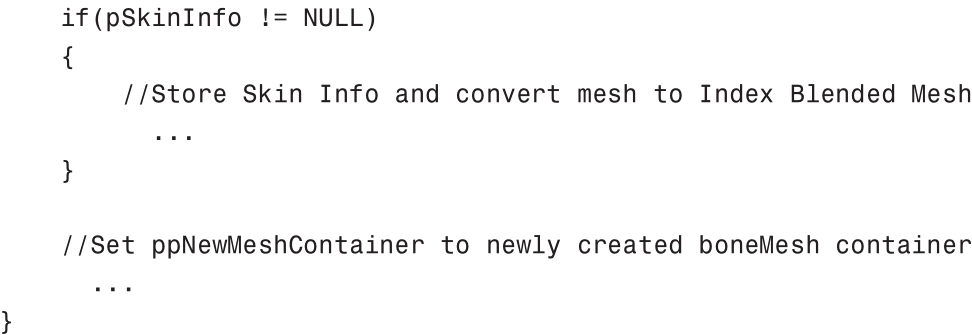


如您所见，机器人手臂的每个部分都是刚性的，因此不需要蒙皮。

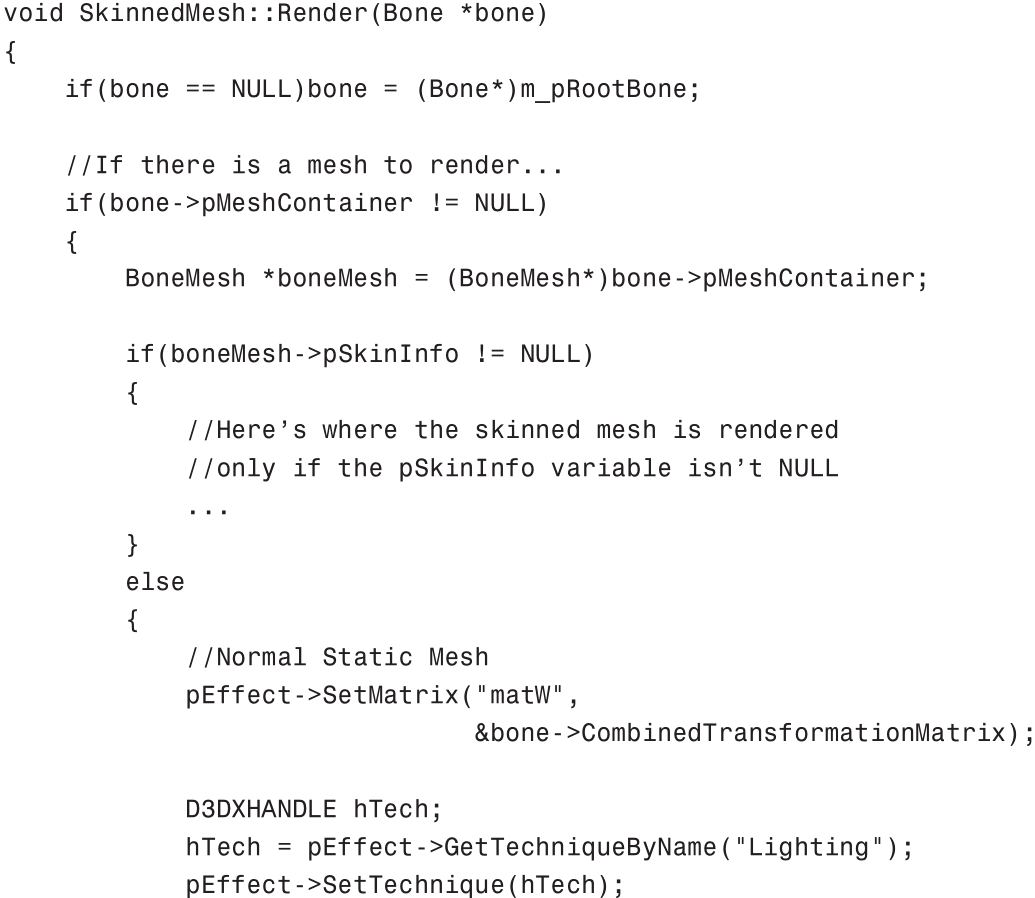
另一种需要刚性/实体对象的情况是它们与蒙皮网格结合使用时。在之前的士兵蒙皮的例子中，你可能已经注意到他缺少头盔和步枪。那是因为这两个对象是刚性对象，不包含蒙皮信息。包含这些对象的一种方法是将它们中的所有顶点分配给一个骨骼（头骨，例如，在头盔的情况下）。但是，这将严重浪费CPU / GPU功耗。

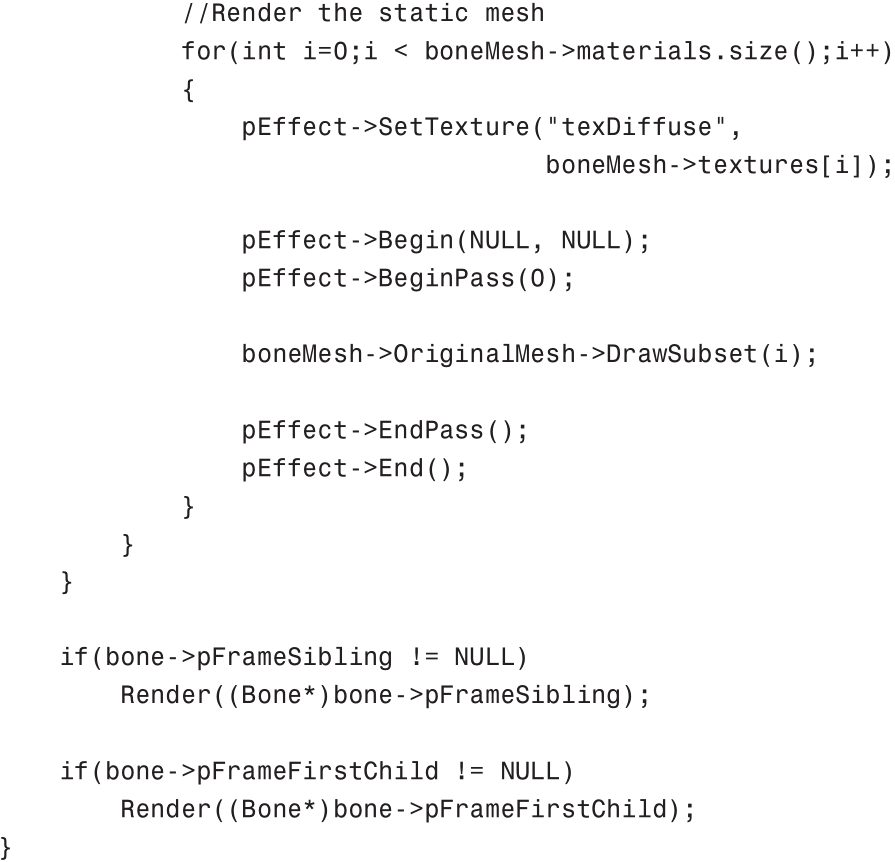
在本节中，您将学习如何从相同的.x加载和渲染蒙皮网格和静态网格物体。不过，坦率地说，本书已经介绍了加载过程。在CreateMeshContainer()函数中加载网格实际上已经完成。这里是这个函数的另一个高级功能:



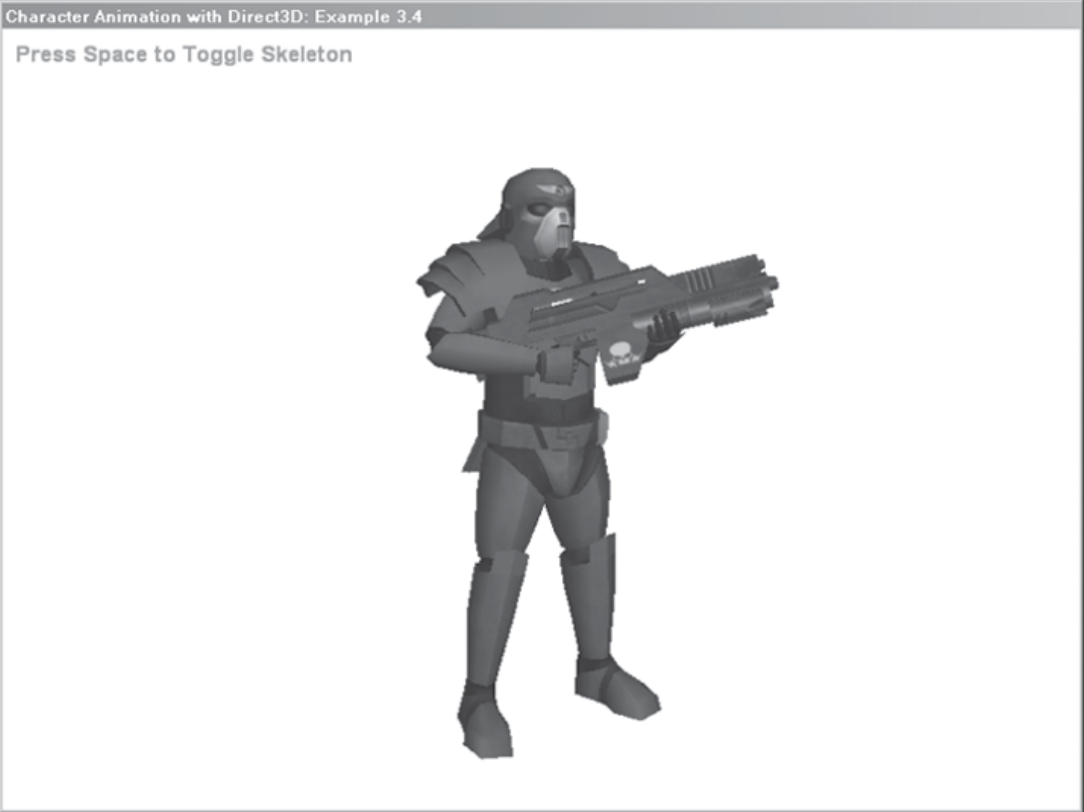


如您所见，如果此函数的pSkinInfo参数不为NULL，则只将网格转换为索引混合网格。但是对于士兵的头盔和步枪，pSkinInfo参数当然是NULL，因此网格不会被转换。但是，网格数据和所属材料和纹理仍然被复制。所以你真正需要做的就是渲染它们！要做到这一点，您只需要将静态网格物体的渲染添加到SkinnedMesh :: Render（）函数中。





静态网格物体仍然锁定在骨骼层次结构中。 如您所见，在设置静态网格物体的世界矩阵时，可以使用网格链接到的骨骼的组合变换矩阵。因此，当您为角色的颈骨设置动画时，头盔将自动跟随。现在，您可以使用此代码渲染完全没有蒙皮部分的机器人角色，或者像具有蒙皮和静态网格物体的士兵一样的混合角色。



恭喜你!如果你还在阅读，你已经读完了整本书最丰富的一章。希望你能从中学到一些东西。将一些顶点附加到骨架上是一个漫长的过程，不是吗？

在本章的最后，你没有比第2章更多的东西来展示你的工作。说实话，在下一章中你将真正体验到收益 - 当你为骨架设置动画时（以及它的角色）。

花点时间再看看每个例子;您将会从这些代码中学到很多东西。

有时间的话可以试试一下操作。

实现自己的Skinned Mesh类，并让它同时支持硬件蒙皮和软件蒙皮。

查看软件蒙皮示例中人物阴影的实现。使用硬件蒙皮来同样实现这个角色。

如果您可以访问3D建模软件，请创建一个蒙皮角色，将其导出为.x文件格式，然后将其读入您的应用程序。

研究SkinnedMesh类中的RenderSkeleton（）函数。试着可视化哪个附有BoneMesh的骨骼。

实现您自己的Bone，BoneMesh和BoneHierarchyLoader类版本。将新成员添加到您在自己的CreateMeshContainer（）函数中初始化的这些类中。