* [Advanced](http://unity3d.com/support/documentation/Manual/Advanced.html) 高级
  + [Profiler](http://unity3d.com/support/documentation/Manual/Profiler.html) 分析器
  + [Loading Resources at Runtime](http://unity3d.com/support/documentation/Manual/Loading%20Resources%20at%20Runtime.html)在运行时装入资源
  + [Modifying Source Assets Through Scripting](http://unity3d.com/support/documentation/Manual/Modifying%20Source%20Assets%20Through%20Scripting.html)通过脚本修改源资产
  + [Update Order](http://unity3d.com/support/documentation/Manual/Update%20Order.html) 更新指令
  + [Shadows in Unity](http://unity3d.com/support/documentation/Manual/Shadows.html) Unity中的阴影
    - [Directional Shadow Details](http://unity3d.com/support/documentation/Manual/DirectionalShadowDetails.html)方向阴影细节
    - [Troubleshooting Shadows](http://unity3d.com/support/documentation/Manual/Shadow%20Troubleshooting.html)阴影疑难解答
    - [Shadow Size Computation](http://unity3d.com/support/documentation/Manual/Shadow%20Size%20Details.html)阴影大小计算
  + [Optimizing Graphics Performance](http://unity3d.com/support/documentation/Manual/Optimizing%20Graphics%20Performance.html)优化图形性能
    - [Modeling Optimized Characters](http://unity3d.com/support/documentation/Manual/Modeling%20Optimized%20Characters.html)模型优化人物
    - [Optimizing for integrated graphics cards](http://unity3d.com/support/documentation/Manual/OptimizeForIntegratedCards.html)优化集成图形卡
    - [Rendering Statistics Window](http://unity3d.com/support/documentation/Manual/RenderingStatistics.html)渲染统计窗口
  + [Reducing File Size](http://unity3d.com/support/documentation/Manual/Reducing%20File%20size.html)减小文件大小
  + [Web Player Streaming](http://unity3d.com/support/documentation/Manual/Web%20Player%20Streaming.html) WEB播放器流
  + [Web Player Deployment](http://unity3d.com/support/documentation/Manual/Web%20Player%20Deployment.html) WEB播放器部署
    - [HTML code to load Unity content](http://unity3d.com/support/documentation/Manual/HTML%20code%20to%20load%20Unity%20Web%20Player%20content.html) HTML代码装入Unity内容
    - [Customizing the Unity Web Player loading screen](http://unity3d.com/support/documentation/Manual/Customizing%20the%20Unity%20Web%20Player%20loading%20screen.html)自定义Unity WEB播放器装入屏幕
    - [Customizing the Unity Web Player's Behavior](http://unity3d.com/support/documentation/Manual/WebPlayerBehaviorTags.html)自定义Unity WEB播放器行为
    - [Unity Web Player and browser communication](http://unity3d.com/support/documentation/Manual/Unity%20Web%20Player%20and%20browser%20communication.html) Unity WEB播放器和浏览器通信
    - [Detecting the Unity Web Player using browser scripting](http://unity3d.com/support/documentation/Manual/Detecting%20the%20Unity%20Web%20Player%20using%20browser%20scripting.html)使用浏览脚本检测Unity WEB播放器
    - [Publishing active content](http://unity3d.com/support/documentation/Manual/Publishing%20active%20content.html)发布活动内容
  + [Web Player Debugging](http://unity3d.com/support/documentation/Manual/Web%20Player%20Debugging.html) WEB播放器调试
  + [Plugins - Pro only feature](http://unity3d.com/support/documentation/Manual/Plugins.html)插件—专业版唯一功能
  + [Build Player Pipeline](http://unity3d.com/support/documentation/Manual/Build%20Player%20Pipeline.html)编译播放器管道
  + [Command line arguments](http://unity3d.com/support/documentation/Manual/Command%20Line%20Arguments.html)命令行参数
  + [Shaders](http://unity3d.com/support/documentation/Manual/Shaders.html)阴影
    - [Shaders: Getting started](http://unity3d.com/support/documentation/Manual/ShaderTut1.html)阴影：开始
    - [Shaders: Vertex and Fragment Programs](http://unity3d.com/support/documentation/Manual/ShaderTut2.html)阴影：顶点和片段程序
  + [Graphics Emulation](http://unity3d.com/support/documentation/Manual/GraphicsEmulation.html)图形仿真
  + [Network Emulation](http://unity3d.com/support/documentation/Manual/NetworkEmulation.html)网络仿真
  + [Visual Studio C# Integration](http://unity3d.com/support/documentation/Manual/VisualStudioIntegration.html) Visual Studio C#集成
  + [Using External Version Control Systems with Unity](http://unity3d.com/support/documentation/Manual/ExternalVersionControlSystemSupport.html) 使用Unity的扩展版本控制系统

[**Unity Manual**](http://unity3d.com/support/documentation/Manual/index.html) > [**Advanced**](http://unity3d.com/support/documentation/Manual/Advanced.html) > [**Profiler**](http://unity3d.com/support/documentation/Manual/Profiler.html)

Unity手册->高级->分析器

**Profiler分析器**

The Unity Profiler helps you to optimize your game. It reports for you how much time is spent in the various areas of your game. For example, it can report the percentage of time spent rendering, animating or in your game logic.

Untiy分析器帮助你优化你的游戏。它为你报告花费多少时间在你的游戏各个领域里。例如，它可以报告花费的时间百分比用于渲染，动画，或在你的游戏逻辑。

The profiler was added at Unity version 2.6. It is a Unity Pro feature. 分析器是加在Unity版本2.6里。这是一个Unity专业版的功能。

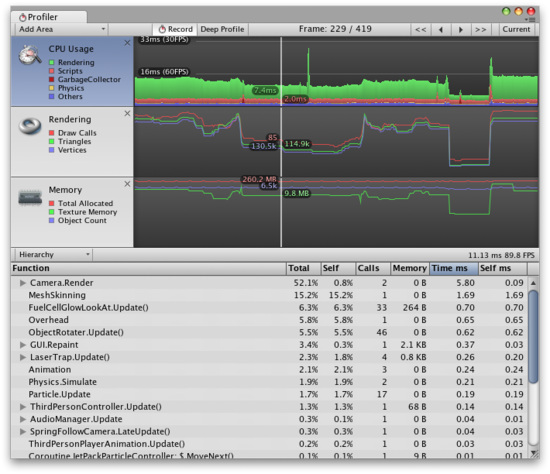
You can play your game in the Editor with Profiling on, and it will record performance data. The Profiler window then displays the data in a timeline, so you can see the frames or areas that spike (take more time) than others. By clicking anywhere in the timeline, the bottom section of the Profiler window will display detailed information for the selected frame.

你可以在编辑器里播放你的游戏并开启分析，它将记录性能数据。 分析器窗口然后在一个时间线里显示数据，因此可以看到框架或区域，峰值（需要更多的时间）超过其它的。按一下在任何时间，该事件分析器窗口底部的部分将显示选定的帧的详细信息。

Note that profiling has to *instrument* your code. This instrumentation has a small impact on the performance of your game. Typically this overhead is small enough to not affect the game framerate. When using profiling it is typical to consider only the ratio (or percentage) of time spent in certain areas. Also, to improve performance focus on those parts of the game that consume the most time. Compare profiling results before and after code changes and determine the improvements you measure. Sometimes changes you make to improve performance might have a negative effect on frame rate; unexpected consequences of code optimization should be expected.

请注意，要分析器必须分析你的代码。这种仪器拥有对你的游戏的性能影响很小。一般来说，这个开销是足够小，不会影响游戏的帧率。当使用分析它通常只考虑比率（或百分比在某些区域所花的时间）。此外，为了提高性能对游戏的部分消耗最多的焦点。比较前后修改代码，并确定你的改进措施以及分析结果。有时会改变你对提高性能可能会对帧速率的产生负面效应;代码优化意想不到的后果应被预期。

*Profiler window 分析器窗口*



**Profiler Controls 分析器控制**



Profiler controls are in the toolbar at the top of the window. Use these to turn profiling on and off, navigate through profiled frames and so on. The transport controls are at the far right end of the toolbar. Note that when the game is running and the profiler is collecting data clicking on any of these transport controls will pause the game. The controls go to the first recorded frame, step one frame back, step one frame forward and go to the last frame respectively. The profiler does not keep all recorded frames, so the notion of the *first* frame should really be though of as the oldest frame that is still kept in memory. The "current" transport button causes the profile statistics window to display data collected in real-time.

分析器控制在工具栏在窗口的顶部。使用这些分析打开和关闭，浏览分析框架等。传输控制在工具栏的最右侧。请注意，当游戏运行和分析器正在收集数据，点击任何这些传输控制将暂停游戏。这些控制转到第一个记录的框架，一步一帧回来，一步一帧前进到各自的最后一帧。分析器并没有记录全部的帧，所以第一帧的概念真正应该作为最原始的框架仍然保存在内存中。 “当前”传输按钮导致统计分析窗口去显示的实时收集的数据。

**Deep Profiling 深度分析**

When you turn on **Deep Profile**, *all* your script code is profiled - that is, all function calls are recorded. This is useful to know where exactly time is spent in your game code.

当你打开深度分析，所有的脚本代码被分析--也就是说，所有的函数调用记录。这用于去了解准确时间应用到何处在你的游戏代码里。

Note that Deep Profiling incurs a **very large overhead** and uses a lot of memory, and as a result your game will run significantly slower while profiling. If you are using complex script code, Deep Profiling might not be possible at all. Deep profiling should work fast enough for small games with simple scripting. If you find that Deep Profiling for your entire game causes the frame rate to drop so much that the game barely runs, you should consider not using this approach, and instead use the approach described below. You may find deep profiling more helpful as you are designing your game and deciding how to best implement key features. Note that for large games deep profiling may cause Unity to run out of memory and so for this reason deep profiling may not be possible.

请注意，深度分析导致一个非常大的开销，使用了大量的内存，因此你的游戏运行显著降低在分析过程中。如果你正在使用复杂的脚本代码，深度分析可能无法分析所有。深度分析应工作足够快对于简单的脚本小游戏。如果你发现为你的整整个游戏深度分析导致刚刚运行的游戏帧速率下降很多，你应该考虑不使用这种方法，而是使用如下介绍的方法。你可能会发现深度的分析提供更多帮助在你设计你的游戏时，并决定如何最好地实现主要功能。请注意，对于大型游戏的深度分析可能导致Unity的运行内存不足，为此深度分析可能无法接受。

Manually profiling blocks of your script code will have a smaller overhead than using Deep Profiling. Use [Profiler.BeginSample](http://unity3d.com/support/documentation/ScriptReference/Profiler.BeginSample.html) and [Profiler.EndSample](http://unity3d.com/support/documentation/ScriptReference/Profiler.EndSample.html) scripting functions to enable and disable profiling around sections of code.

手工分析你的脚本代码块将有一个使用小的开销与使用深度分析相比。使用Profiler.BeginSample和Profiler.EndSample脚本函数启用或禁止分析代码段。

**Profiler Timeline 时间线分析器**



The upper part of the Profiler window displays performance data over time. When you run a game, data is recorded each frame, and the history of the last several hundred frames is displayed. Clicking on a particular frame will display it's details in the lower part of the window. Different details are displayed depending on which timeline area is currently selected.

在分析器窗口的上部中显示随着时间的推移性能数据。当你运行一个游戏，数据记录每一帧，并显示过去几百帧的历史。点击某一帧将在窗口的下半部分显示它的细节。不同细节的显示取决于时间区的当前选择。

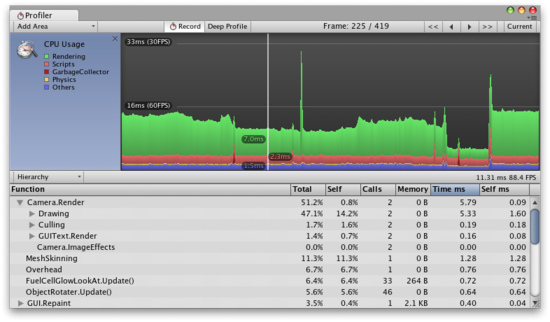
The vertical scale of the timeline is managed automatically and will attempt to fill the vertical space of the window. Note that to get more detail in say the CPU Usage area you can remove the Memory and Rendering areas. Also, the splitter between the timeline and the statistics area can be selected and dragged downward to increase the screen area used for the timeline chart.

时间线的纵坐标是自动管理并尝试填补窗口的垂直空间。请注意，要获得更详细地在你可以删除内存和渲染领域的CPU使用率区域中说明。此外，在时间线和统计领域之间的分析器可以被选择和向下拖动以增加屏幕面积的时间表图表的使用。

The timeline consists of several areas: CPU Usage, Rendering and Memory. These areas can be removed by clicking the close button in the panel, and re-added again using the Add Area drop down in the Profile Controls bar.

时间线包括几个方面：CPU使用率，渲染和内存。这些领域可以移除通过单击面板关闭按钮，并重新添加再次使用新增面积下降在分析控制栏里。

**CPU Usage Area CPU利用率**



The CPU Usage area displays where time is spent in your game. When it is selected, the lower pane displays hierarchical time data for the selected frame.

CPU使用率区域显示在你的游戏哪里的时间是被用过的。当它被选中，下部窗格显示分层的实时数据。

* Hierarchy mode - displays hierarchical time data. 层次模式—显示层次时间数据
* Group Hierarchy mode - groups time data into logical groups (Rendering, Physics, Scripts etc.).
* 层次群模式—逻辑组里的群时间数据（渲染、物理、脚本等）。

Because children of any group can be in different group (e.g. some script might call rendering functions), the percentages of group times often add up to more than 100%. (This is not a bug.)

因为任何组的可在不同的组里（例如：一些脚本也许调用渲染函数），组时间的百分比常常总计超过100%（这是一个BUG）。

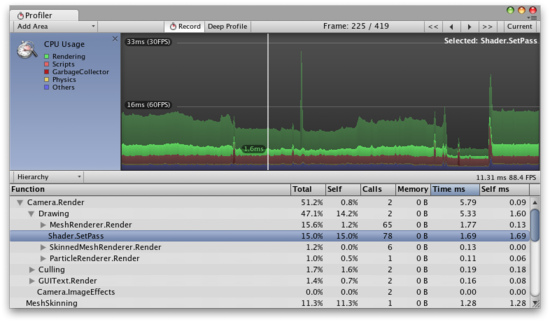
The way the CPU chart is stacked can be reordered by simply dragging chart labels up & down.

在CPU的图表方法是堆积可以通过简单的拖动图表标签升降排序。

When an item it selected in the lower pane, it's contribution to the CPU chart is highlighted (and the rest are dimmed). Clicking on an item again de-selects it.

当一个项目是在较低的窗格中选中，它的贡献是突出的CPU图表（并变灰）。在一个项目再次点击取消选择它。

*Shader.SetPass is selected and it's contribution is highlighted in the chart.*

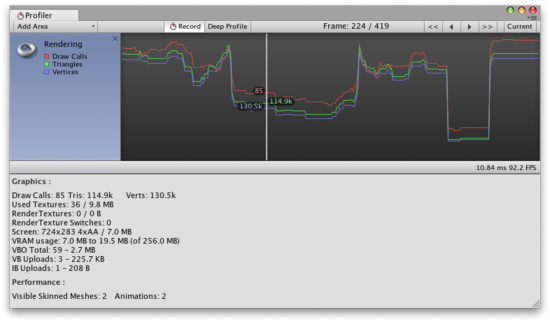


*Shader.SetPass被选中，它的贡献是在图表中突出显示。*

In the hierarchical time data the self time refers to the amount of time spent in a particular function not including the time spent calling sub-functions. In the screenshot above, for example 51.2% of time is spent in the Camera.Render function. This function does a lot of work and calls the various drawing and culling functions. Excluding all these functions only 0.8% of time is spent actually in the Camera.Render function.

在分层时间数据的自我时间是指在某一特定功能所用的时间不包括调用子功能所花费的时间。在上面的截图里，例如51.2％的时间，花费在Camera.Render函数上。这个函数做了大量工作并调用若干绘画函数和精选函数。除去所有这些只有0.8％的时间花费在Camera.Render函数实际功能上。

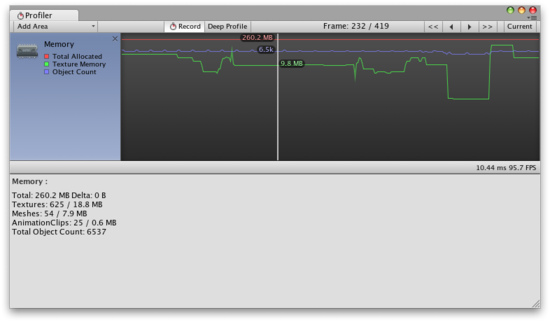
**Rendering Area 渲染区域**



The Rendering area displays rendering statistics. The Number of Draw Calls, Triangles and Vertices rendered is displayed graphical in the timeline. The Lower pane displays more rendering statistics and these more closely match the ones shown in the GameView [Rendering Statistics](http://unity3d.com/support/documentation/Manual/RenderingStatistics.html) window.

渲染区域显示的渲染统计。绘制调用的数量，三角形和顶点渲染是在时间线中显示图形。下部窗格显示更多的渲染统计而这些更紧密的配合显示在GameView的渲染统计窗口。

**Memory Area 内存区域**



The Memory area displays some memory usage data: 内存区域显示内存使用的一些数据：

**Total Allocated** is the total RAM used by the application. Note that in the Unity Editor this is memory used by everything in the editor; game builds will use much less.

总分配内存是应用程序使用内存的合计。请注意，在Unity的编辑器，这是在编辑器中所使用的一切内存;游戏编译使用较少。

**Texture Memory** is the about of video memory used by the textures in the current frame.

纹理内存是关于纹理使用的显存在当前游戏里。

**Object Count** is the total number of Objects that are created. If this number rises over time then it means your game is creating some objects that are never destroyed.

对象计数是创建的对象总数。如果这个数字随着时间的推移上升那么这意味着你的游戏的创建的一些对象从未释放过。

**See Also参见**

* [Optimizing Graphics Performance](http://unity3d.com/support/documentation/Manual/Optimizing%20Graphics%20Performance.html) page.优化的图形性能页。

[**Unity Manual**](http://unity3d.com/support/documentation/Manual/index.html) > [**Advanced**](http://unity3d.com/support/documentation/Manual/Advanced.html) > [**Loading Resources at Runtime**](http://unity3d.com/support/documentation/Manual/Loading%20Resources%20at%20Runtime.html)

Unity手册->高级->在运行时装入资源

# Loading Resources at Runtime 在运行时装入资源

There are a number of scenarios where you want to include some kind of asset in your finished game that is not included inside the primary game executable; standalone or web player. One scenario is to slim a webplayer download size, and dynamically download and instantiate individual objects or items. This way only the relevant and needed objects are used by the end-user. Another scenario might be for downloadable content like additional weapons, environments, characters, or even full levels.

有多种方案，在你完成的游戏里的包含多种类型的资产但不包含于可执行基本游戏内部：独立或网络播放器。一种情况是小的webplayer下载尺寸，以及动态下载和实例化对象的项目或它自己。这只是相关的和需要的对象被最终用户使用。另一种方案可能会像其他的武器，环境，人物，甚至完全级别下载内容。

In Unity Pro, you can choose between two options for accomplishing these tasks: Asset Bundles, and Resource Folders. Unity Indie license holders can use Resource Folders only.

在Unity的专业版本里，你可以选择两个选项为完成这些任务：资产包，和资源文件夹。Unity独立的许可以仅能使用的资源文件夹。

## Asset Bundles (Unity Pro-only) 资产包（Uniyt 仅专业版具有）

An Asset Bundle is an external collection of assets. You can have many Asset Bundles and therefore many different external collections of assets. These files exist outside of the built Unity player, usually sitting on a web server for end-users to access dynamically.

一个资产包是资产的外部集合。你可以有很多资产包，因此许多不同资产的外部集合。这些文件存在创建的Unity的 播放器的外面，通常是位于一个Web服务器上为用户访问动态。

To build an Asset Bundle, you call [BuildPipeline.BuildAssetBundle()](http://unity3d.com/support/documentation/ScriptReference/BuildPipeline.BuildAssetBundle.html) from inside a script. In the arguments, you specify an array of Objects to be included in the built file, along with some other options. This will build a file that you can later load dynamically in the runtime by using [AssetBundle.Load()](http://unity3d.com/support/documentation/ScriptReference/AssetBundle.Load.html).

建立一个资产包，你调用BuildPipeline.BuildAssetBundle（）从内部一个脚本里。在变量里，你指定一个对象数组包被含在建立的文件里，以及一些其他的选择。这将建立一个文件，你以后可以在运行时动态加载使用AssetBundle.Load（）。

## Resource Folders (Indie and Pro) 资源文件夹（独立版本和专业版本）

Resource Folders are collections of assets that are included in the built Unity player, but are not necessarily linked to any GameObject in the Inspector.

资源文件夹是资产的集合，包括，在Unity里所建立的播放器，但不必连接到任何在检视器里的GameObject。

To put anything into a Resource Folder, you simply create a new folder inside the Project View, and name the folder "Resources". You can have multiple Resource Folders organized differently in your Project. Whenever you want to load an asset from one of these folders, you call [Resources.Load()](http://unity3d.com/support/documentation/ScriptReference/Resources.Load.html).

放任何东西到一个资源文件夹里，你只需在项目视图里创建一个的新文件夹，并命名文件夹为“资源”。你可以有多个资源文件夹组织不同的资源在你的项目里。每当你要加载的其中一个文件夹的资产，你调用Resources.Load（）。

If your target deployable is a Streaming Web Player, you can define which scene will include everything in your Resource Folders. You do this in the Player Settings, accessible via Edit->Project Settings->Player. Set the First Streamed Level With Resources parameter, and all assets in your Resource Folders will be loaded when this level streams in to the end-user.

如果你的目标是部署一个流Web播放器，你可以定义场景将包含在你资源文件夹里的一切。你在播放机设置做到这一点，通过访问Edit->Project Settings->Player。设置第一流级使用资源参数，并且在你的资源文件夹的所有资产将被最终用户装载在这一水平流。

[**Unity Manual**](http://unity3d.com/support/documentation/Manual/index.html) > [**Advanced**](http://unity3d.com/support/documentation/Manual/Advanced.html) > [**Modifying Source Assets Through Scripting**](http://unity3d.com/support/documentation/Manual/Modifying%20Source%20Assets%20Through%20Scripting.html)

Unity手册->高级->通过脚本修改源资产

**Modifying Source Assets Through Scripting通过脚本修改源资产**

**Automatic Instantiation自动实例**

Usually when you want to make a modification to any sort of game asset, you want it to happen at runtime and you want it to be temporary. For example, if your character picks up an invincibility power-up, you might want to change the **shader** of the **material** for the player character to visually demonstrate the invincible state. This action involves modifying the material that's being used. This modification is not permanent because we don't want the material to have a different shader when we exit **Play Mode**.

通常当你想做一个任何游戏资产排序的修改，你希望它在运行时发生并希望它是临时的。例如，如果你的角色拿起一不可战胜的权力，你可能想改变玩家角色的物质着色直观显示战无不胜的状态。这一行动涉及到修改，目前已被使用的材料。这种修改是不是永久性的，因为我们不想材料有不同的着色，当我们离开播放模式时。

However, it is possible in Unity to write scripts that will permanently modify a source asset. Let's use the above material example as a starting point.

然而，它有可能在Unity编写脚本去永久地修改源代码的资产。让我们以上述材料的例子为起点。

To temporarily change the material's shader, we change the **shader** property of the **material** component.

要暂时改变材料的着色，我们改变材料的组件部分阴影的属性。

private var invincibleShader = Shader.Find ("Specular");

function StartInvincibility {

[renderer](http://unity3d.com/support/documentation/ScriptReference/Renderer.html).[material](http://unity3d.com/support/documentation/ScriptReference/Renderer-material.html).shader = invincibleShader;

}

When using this script and exiting Play Mode, the state of the [**material**](http://unity3d.com/support/documentation/ScriptReference/Material.html) will be reset to whatever it was before entering Play Mode initially. This happens because whenever renderer.material is accessed, the material is automatically instantiated and the instance is returned. This instance is simultaneously and automatically applied to the renderer. So you can make any changes that your heart desires without fear of permanence.

当使用脚本并退出播放模式，材料的状态将返回到它最初播放之前的状态。这个发生因为无论何时渲染器、材质被访问，材质自动实例并且实例将返回。这个实例是同时的和自动的应用于渲染器。因此你可做一些你心里期望的没有永久害怕的更改。

**Direct Modification 直接修改**

**IMPORTANT NOTE 重要注释**

The method presented below will modify actual source asset files used within Unity. These modifications are not undoable. Use them with caution.

下面将修改文件的实际来源的资产使用的在Unity范围内提出的方法。这样的修改是不可撤消的。请小心使用他们。

Now let's say that we don't want the material to reset when we exit play mode. For this, you can use [renderer.sharedMaterial](http://unity3d.com/support/documentation/ScriptReference/Renderer-sharedMaterial.html). The sharedMaterial property will return the actual asset used by this renderer (and maybe others).

现在让我们说，我们不希望这些材料在我们退出播放模式后重置。为此，你可以使用renderer.sharedMaterial。该sharedMaterial属性将返回实际的资产这个渲染器（或者其他人）使用。

The code below will permanently change the material to use the Specular shader. It will not reset the material to the state it was in before Play Mode.

下面的代码将永久改变材料使用的反射阴影。它不会重置材料中的状态是在播放模式之前。

private var invincibleShader = Shader.Find ("Specular");

function StartInvincibility {

[renderer](http://unity3d.com/support/documentation/ScriptReference/Renderer.html).[sharedMaterial](http://unity3d.com/support/documentation/ScriptReference/Renderer-sharedMaterial.html).shader = invincibleShader;

}

As you can see, making any changes to a sharedMaterial can be both useful and risky. Any change made to a sharedMaterial will be permanent, and not undoable.

正如你所看到的，做任何更改sharedMaterial可以是有益的和危险的。任何改动的sharedMaterial将是永久的和不可撤消的。

**Applicable Class Members 可用的类成员**

The same formula described above can be applied to more than just materials. The full list of assets that follow this convention is as follows:

在上述相同的准则可应用于不仅仅是材料。资产的完整列表，按照约定如下：

* Materials: renderer.material and renderer.sharedMaterial
* 材料renderer.material 和 renderer.sharedMaterial
* Meshes: meshFilter.mesh and meshFilter.sharedMesh
* 网格：meshFilter.mesh 和 meshFilter.sharedMesh
* Physic Materials: collider.material and collider.sharedMaterial
* 物理材料：collider.material 和 collider.sharedMaterial

**Direct Assignment 直接赋值**

If you declare a public variable of any above class: Material, Mesh, or Physic Material, and make modifications to the asset using that variable instead of using the relevant class member, you will not receive the benefits of automatic instantiation before the modifications are applied.

如果你声明任何上述类的公共变量：材料，网格，或物理材料，并作出资产使用该变量而不是使用有关的类成员的修改，你将不会收到自动实例化的好处在修改被应用之前。

**Assets that are not automatically instantiated 资产不自动化实例**

There are two different assets that are never automatically instantiated when modifying them.

有两个不同的资产永远不会自动实例当修改它们时。

* [Texture2D](http://unity3d.com/support/documentation/ScriptReference/Texture2D.html) 2D纹理
* [TerrainData](http://unity3d.com/support/documentation/ScriptReference/TerrainData.html) 地形数据

Any modifications made to these assets through scripting are always permanent, and never undoable. So if you're changing your terrain's heightmap through scripting, you'll need to account for instantiating and assigning values on your own. Same goes for Textures. If you change the pixels of a texture file, the change is permanent.

通过脚本对这些资产的任何修改总是永久的，永不撤消的。因此，如果你通过脚本更改你的地形的高度图，你需要考虑实例和分配你自己的值。纹理也是一样的。如果你更改了文件的纹理的像素，变化是永久性的。

[**Unity Manual**](http://unity3d.com/support/documentation/Manual/index.html) > [**Advanced**](http://unity3d.com/support/documentation/Manual/Advanced.html) > [**Update Order**](http://unity3d.com/support/documentation/Manual/Update%20Order.html)

Unity手册->高级->更新指令

# Update Order 更新指令

When you're keeping track of game logic and interactions, animations, camera positions, etc., there are a few different events you can use. The most common is to perform everything inside the Update() function. This is great for most kinds of operations, but there are other options! 当你保持游戏逻辑和互动，动画，摄像机位置等，有几个不同的事件可以使用。最常见的是执行内部更新Update()函数。这对于大多数种类的运算是很大的，但也有其他的选择！

## FixedUpdate 固定更新

**FixedUpdate**() is often called more frequently than Update(). It can be called multiple times per frame, if frame rate is low; and it can be not called between frames at all if frame rate is high. All Physics calculations and updates occur immediately before **FixedUpdate**(). When applying movement calculations inside **FixedUpdate**(), you do not need to multiply your values by Time.deltaTime. This is because **FixedUpdate**() is called on a reliable timer, independent of the frame rate.

FixedUpdate()通常调用的次数超过Update()。它可以被每帧多次调用，如果帧速率低，而且可以在帧与所有帧速是高之间不调用。所有的物理计算和更新发生FixedUpdate()之前。当内部FixedUpdate()的应用移动计算，你不需要你的值乘以Time.deltaTime。这是因为FixedUpdate()被称为一个可靠的计时器，不依赖与帧速率。

## Update 更新

Update() is called once per frame. It is used far more commonly than any other option from a user perspective.

Update()是被每帧调用一次。它用于更为普遍比任何从用户的角度任何其它选择。

## LateUpdate 最后更新

**LateUpdate**() is called once per frame, after Update() has finished. Any calculations that are performed in Update() will have completed when **LateUpdate**() begins. A common usage for **LateUpdate**() would be a following third-person camera. If you make your character move and turn inside Update(), you can perform all camera pointing and moving calculations in **LateUpdate**(). This will ensure that the character has moved completely before the camera tries to point itself at him.

LateUpdate（）每帧调用一次，在Update()已经完成后。任何在最新的计算结果表明Update()将完成时LateUpdate（）开始。一个LateUpdate（）共同使用将是下面的第三人称摄像机。如果你把你的角色移动和转换成Update()内部，你可以执行所有的摄像机指向在LateUpdate（）里移动计算。这将确保角色完全移动在摄像机尝试指定它自己之前。

## Coroutine 协同

Coroutines are begun by using **StartCoroutine**(). Within the Coroutine, any time yield is called, the Coroutine will stop and resume where it left off after LateUpdate**()** has completed.

协同通过使用StartCoroutine()开始。在协同内部，任何时间段被调用，协同将停止并恢复在调用LateUpdate（）已完成后它离开哪里。

[**Unity Manual**](http://unity3d.com/support/documentation/Manual/index.html) > [**Advanced**](http://unity3d.com/support/documentation/Manual/Advanced.html) > [**Shadows in Unity**](http://unity3d.com/support/documentation/Manual/Shadows.html)

Unity手册->高级->Unity中的阴影

**Shadows in Unity Unity中的阴影**

Unity 2.0 Pro makes it possible to use real-time **shadows** on any light. Objects can cast shadows onto each other and onto parts of themselves ("self shadowing"). All types of [Lights](http://unity3d.com/support/documentation/Components/class-Light.html) - Directional, Spot and Point - support shadows. Unity2.0专业版本可能使用任何光线实时阴影。对象可以投射到对方，走上自己的阴影部分（“自我阴影”）。所有类型的光源-方向光源，聚光光源和点光源-支持阴影。

Using shadows can be as simple as choosing **Hard Shadows** or **Soft Shadows** on a [Light](http://unity3d.com/support/documentation/Components/class-Light.html). However, if you want optimal shadow quality and performance, there are some additional things to consider.

使用阴影可以很简单的选择上硬阴影或软阴影。不过，如果你想最佳的阴影质量和性能，还有一些额外的事情要考虑。

The [Shadow Troubleshooting](http://unity3d.com/support/documentation/Manual/Shadow%20Troubleshooting.html) page contains solutions to common shadowing problems. 阴影疑难解答页包含解决公共问题的阴影。

Curiously enough, the best shadows are non-realtime ones! Whenever your game level geometry and lighting is static, just precompute lightmaps in your 3D application. Computing shadows offline will always result in better quality and performance than displaying them in real time. *Now onto the realtime ones...*

奇怪的是，最好的阴影是非实时的！每当你的游戏水平几何和照明是静态的，只要预先计算在您的3D应用光映射。离线阴影计算总是产生更好的质量和性能超过实时显示它们。现在，到实时的...

**Tweaking shadow quality 调整阴影质量**

Unity uses so called [shadow maps](http://en.wikipedia.org/wiki/Shadow_mapping) to display shadows. Shadow mapping is a texture based approach, it's easiest to think of it as "shadow textures" projecting out from lights onto the scene. Thus much like regular texturing, quality of shadow mapping mostly depends on two factors:

Unity使用所谓的阴影图形显示阴影。阴影映射是一种基于纹理的方法，最简单的把它看成“阴影纹理”从光源里投射到场景上。因此，就像常规纹理，阴影映射的质量主要取决于两个因素：

* The **resolution** (size) of the shadow maps. The larger the shadow maps, the better the shadow quality. 该阴影图形分辨率（大小）。较大的阴影图形，更好的阴影质量。
* The **filtering** of the shadows. **Hard shadows** take the nearest shadow map pixel. **Soft shadows** average several shadow map pixels, resulting in smoother looking shadows (but soft shadows are more expensive to render). 过滤的阴影。硬阴影采取就近的阴影图形像素。软阴影平均几个阴影图形像素，阴影看起来平滑的结果（但软阴影的渲染比较昂贵）。

Different **Light** types use different algorithms to calculate shadows. 不同的类型的光源使用不同的算法来计算阴影。

* For Directional lights, the crucial settings for shadow quality are **Shadow Distance** and **Shadow Cascades**, found in [Quality Settings](http://unity3d.com/support/documentation/Components/class-QualitySettings.html). **Shadow Resolution** is also taken into account, but the first thing to try to improve directional shadow quality is reducing shadow distance. All the details about directional light shadows can be found here: [Directional Shadow Details](http://unity3d.com/support/documentation/Manual/DirectionalShadowDetails.html).
* 定向光源，阴影质量的关键设置是阴影距离和阴影级联，在质量设置中找到。阴影清晰度也是考虑，但首先要努力提高定向阴影质量的减少阴影的距离。所有关于定向阴影的细节可以在这里找到：定向阴影细节。
* For Spot and Point lights, **Shadow Resolution** determines shadow map size. Additionally, for lights that cover small area on the screen, smaller shadow map resolutions are used. 对于聚光光源和点光源，阴影清晰度决定阴影映射的大小。此外，对于光源覆盖在屏幕上小的面积，小的阴影图形的清晰度被使用。

Details on how shadow map sizes are computed are in [Shadow Size Details](http://unity3d.com/support/documentation/Manual/Shadow%20Size%20Details.html) page.阴影图形尺寸的详情如何计算在阴影大小详细信息页。

**Shadow performance 阴影性能**

Realtime shadows are quite performance hungry, so use them sparingly. For each light to render its shadows, first any potential shadow casters must be rendered into the shadow map, then all shadow receivers are rendered with the shadow map. This makes shadow casting lights even more expensive than **Pixel lights**, but hey, computers are getting faster as well! 实时阴影要求相当高的性能，所以要适可而止。对于每一个光渲染的它自己的阴影，首先调整任何潜在的阴影都必须进入到阴影图形里，那么所有的阴影接收阴影图形渲染。这使得阴影投影甚至比像素光源昂贵。但是，嘿，电脑速度越来越快的！

**Soft shadows** are more expensive to render than **Hard shadows**. The cost is entirely on the graphics card though (it's only longer shaders), so Hard vs. Soft shadows don't make any impact on the CPU or memory.

软阴影是较昂贵的渲染比硬阴影。费用完全是图形卡，但（这只是长久的阴影），所以硬阴影与软阴影相比，不受任何CPU或内存的影响

[Quality Settings](http://unity3d.com/support/documentation/Components/class-QualitySettings.html) contains a setting called **Shadow Distance** - this is how far from the camera shadows are drawn. Often it makes no sense to calculate and display shadows that are 500 meters away from the camera, so use as low shadow distance as possible for your game. This will help performance (and will improve quality of directional light shadows, see above). 质量设置包含一个名为阴影距离-这就是远距离摄像机阴影绘制。往往是没有意义的计算和显示阴影，有500米距离相机，因此使用尽可能低的阴影距离为你的游戏。这将有助于性能（和将提高定向阴影质量，见上文）。

**Hardware support for shadows 硬件支持的阴影**

Built-in shadows require a fragment program (pixel shader 2.0) capable graphics card. This is the list of supported cards: 内置阴影需要一个片断程序（支持Pixel Shader 2.0）的图形卡。这是支持的卡列表：

* On Windows: windows平台
  + ATI Radeon 9500 and up, Radeon X series, Radeon HD series.
    - When using OpenGL, ATI cards do not support Point light shadows (point lights won't have shadows). 当使用OpenGL，ATI显卡不支持点光源阴影（点光源不会有阴影）。
  + NVIDIA GeForce FX, 6xxx, 7xxx, 8xxx, 9xxx, GeForce GT, GTX series.
  + Intel GMA 3000 (965) and up.
* On Mac OS X: 在Mac平台上
  + ATI Radeon 9500 and up, Radeon X series.
  + NVIDIA GeForce FX, 6xxx, 7xxx, 8xxx, 9xxx series.
    - NVIDIA cards on OS X 10.3 have shadows disabled.
  + Intel GMA 950.
    - Soft shadows are disabled because of driver bugs (hard shadows will be used instead). 软阴影被停用是因为驱动程序错误（硬阴影将代替）。

**Notes 注释**

* Point light shadows are always "hard". Selecting Hard or Soft shadows does not make a difference. 点光源阴影总是“硬”。选择硬阴影或软阴影并没有发挥作用。
* Vertex-lit lights don't have shadows. 顶点背光光源没有阴影。
* Vertex-lit materials won't receive shadows (but do cast shadows). Some other materials don't receive shadows as well (Diffuse Fast). 顶点背光材料将不会收到阴影（但投射阴影）。其它一些材料也不会收到阴影（快速弥漫）。
* Transparent objects don't cast or receive shadows. Transparent Cutout objects do cast and receive shadows. 透明的对象不投射阴影或接收阴影。透明切块物体投射和接收阴影。

[**Unity Manual**](http://unity3d.com/support/documentation/Manual/index.html) > [**Advanced**](http://unity3d.com/support/documentation/Manual/Advanced.html) > [**Shadows in Unity**](http://unity3d.com/support/documentation/Manual/Shadows.html) > [**Directional Shadow Details**](http://unity3d.com/support/documentation/Manual/DirectionalShadowDetails.html)

Unity手册->高级->Unity中的阴影->方向阴影细节

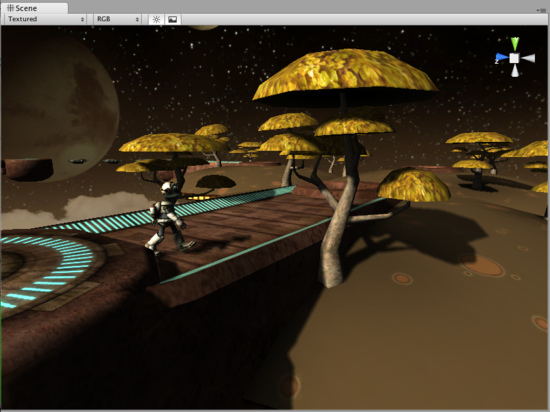
**Directional Shadow Details方向阴影细节**

This page explains [shadows](http://unity3d.com/support/documentation/Manual/Shadows.html) from **Directional** lights in detail. 本从定向光源细节解释阴影。

Directional lights are mostly used as a key light - sunlight or moonlight - in an outdoor game. Viewing distances can be huge, especially in first and third person games, and shadows often require some tuning to get the best quality vs. performance balance for your situation. 定向光源大多是用来作为主光源-阳光或月光-在室外游戏里。视距可以是巨大的，尤其是在第一和第三人称游戏，以及阴影往往需要一些调整，以获得与性能的平衡状况最好的质量。

Let's start out with a good looking shadow setup for a 3rd person game: 让我们开始一个漂亮的阴影设置的第三人称游戏：

*Shadows here look pretty good! 阴影这里看来相当不错！*



Here, visible distance is about 50 game units, so **Shadow Distance** was set to 50 in [Quality Settings](http://unity3d.com/support/documentation/Components/class-QualitySettings.html). Also, **Shadow Cascades** was set to 4, **Shadow Resolution** to High, and the light uses **Soft Shadows**. 在这里，可视距离约50游戏单位，因此在质量设置里设置影子距离为50。此外，阴影级联被设置为4，阴影的分辨率设置为高，而灯光使用软阴影。

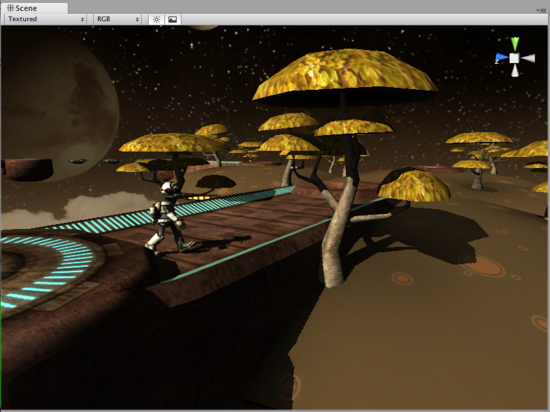
Chapters below dissect each aspect of directional light shadows: 以下各章解剖定向光阴影每个方面：

* [Hard versus Soft shadows](http://unity3d.com/support/documentation/Manual/DirectionalShadowDetails.html#hardvssoft#hardvssoft) 硬阴影与软阴影
* [Shadow Cascade count](http://unity3d.com/support/documentation/Manual/DirectionalShadowDetails.html#cascades#cascades)  阴影级联数
* [Shadow Distance is Important!](http://unity3d.com/support/documentation/Manual/DirectionalShadowDetails.html#distance#distance) 阴影距离很重要！

**Hard versus Soft shadows**硬阴影与软阴影

Using the same light setup, if we switch **Shadow Type** to **Hard Shadows**, then the transition from lit to shadowed regions is "hard" - either something is 100% in shadow, or 100% lit. Hard shadows are faster to render but often they look less realistic. 使用相同的光源设置，如果我们改用阴影类型的硬阴影，然后从照亮阴影区的过渡转换到“硬” -无论事情是100%的阴影，或100％照亮。硬阴影被快速渲染但往往它们看起来不太现实。

*Hard shadows with distance of 50 and four cascades. 50个硬阴影和4个级联距离。*



**Shadow Cascade count 阴影级联数量**

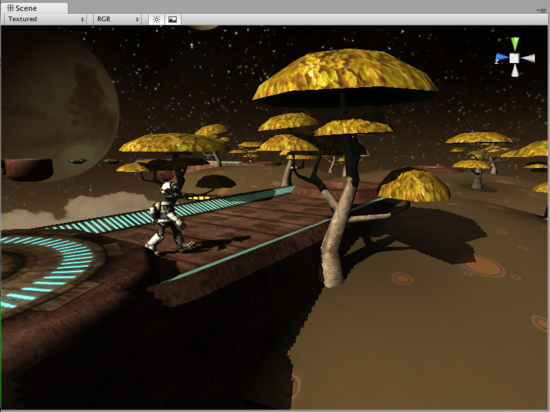
For Directional lights Unity can use so called **Cascaded Shadow Maps** (alternatively called "Parallel Split Shadow Maps") which give very good shadow quality, especially for long viewing distances. Cascaded shadows work by dividing viewing area into progressively larger portions and using the same size shadow map on each. The result is that objects close to the viewer get more shadow map pixels than objects far away.

Unity定向光源可以使用所谓的级联阴影地图（也称“平行拆分阴影地图”），提供很好的阴影质量，特别是长视距。级联分成逐步较大的部分，并在每个使用同样大小的阴影图可视面积阴影工作。其结果是，物体接近观众得到更多的阴影映射像素超过更远的物体。

In the images below we'll use Hard shadows because shadow pixels are better visible there. 在以下图片，我们将使用硬阴影，因为在那里阴影像素更好可见。

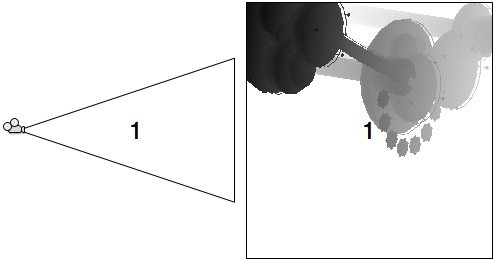
If no cascaded shadow maps were used, the entire shadow distance (still 50 units in our case) must be covered by the shadow texture uniformly. Hard shadows would look like this with no cascades: 如果没有级联阴影地图使用，整个阴影的距离（仍旧50个单位在我们的方案里）必须由阴影纹理均匀覆盖。硬阴影看起来像这样的，没有瀑布：

*Hard shadows with distance of 50 and no cascades. 50距离的硬阴影而没有级联。*



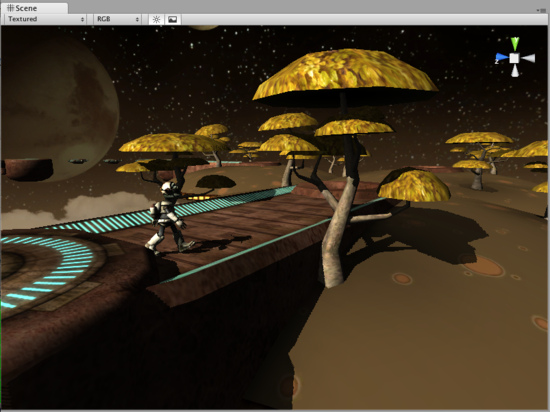
The pixels of the shadow texture are the same size everywhere, and while they look good in distance, the quality is not stellar up close. The shadow texture covers the entire viewing area, and if visualized it would look like this: 阴影纹理像素的大小相同随处可见，而在距离里他们看起来良好，质量是没有星体的近距离。阴影纹理覆盖整个显示区域，如果直观的它将看起来像这样：

*With no cascades, shadow texture covers viewing area uniformly. 没有级联，阴影纹理平均的覆盖。*



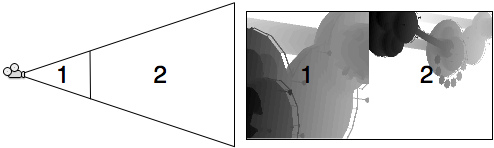
When two shadow cascades are used, the entire shadow distance is divided into a smaller chunk near the viewer and a larger chunk far away. Hard shadows would look like this with two cascades: 当两个阴影级联使用，整个阴影距离分为接近的观众一小块和远处的一大块。硬阴影是这样的两个级联：

*Hard shadows with distance of 50 and two cascades. 50的距离的硬阴影和两个级联。*



In exchange for some performance, we get better shadow resolution up close. 调整一些性能，我们获得更好的阴影的近距离的清晰度。

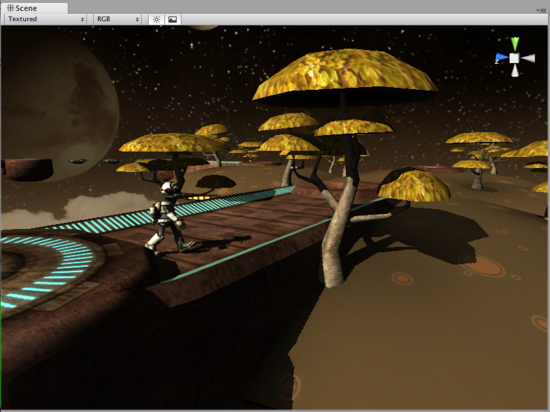
*With two cascades, two shadow textures cover different sized portions of viewing area.*



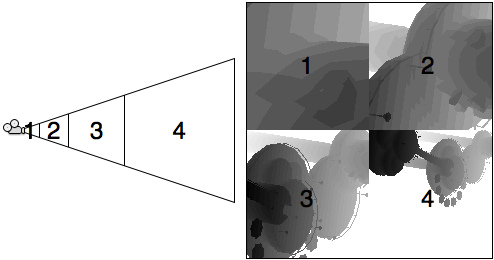
*利用2个级联，2个阴影纹理覆盖不同大小视图区域的位置。*

And finally when four shadow cascades are used, the shadow distance is divided into four progressively larger portions. Hard shadows would look like this with four cascades: 最后，当4级联阴影使用，阴影的距离分为四个大的部分逐渐分化。硬阴影是这样的四个级联：

*Hard shadows with distance of 50 and four cascades. Hey, we've seen this already! 50距离的硬阴影和4个级联。*



*With four cascades, four shadow textures cover different sized portions of viewing area. 利用4个级联，四个阴影纹理覆盖不同大小视图区域位置。*

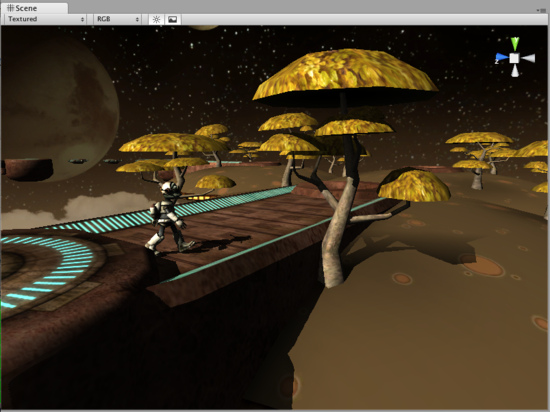


**Shadow Distance is Important!** 阴影距离很重要！

**Shadow Distance** is extremely important for both quality and performance of directional light shadows. Just like shadow cascade count, shadow distance can be set in [Quality Settings](http://unity3d.com/support/documentation/Components/class-QualitySettings.html) and allows an easy way to scale shadows down on less performant hardware. 阴影距离是非常重要的对质量和阴影的定向光性能而言。就像阴影级联计数，在质量设置里可以设定阴影距离，并允许一个简单的方法来伸缩阴影向下在少高性能的硬件上。

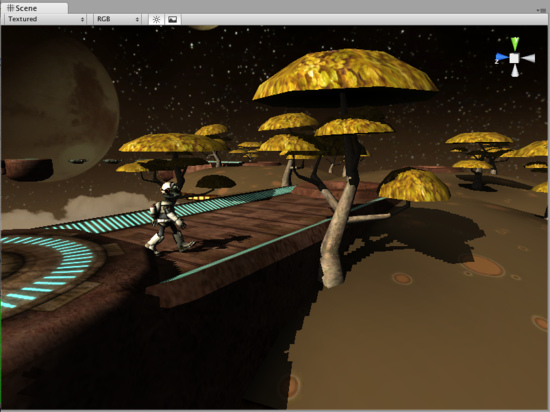
Shadows fade out at the end of shadow distance, and further than that objects are not shadowed. In most situations shadows further than some distance in the game would not be noticeable anyway! 阴影淡出在阴影距离的末尾，并进一步比物体不带阴影。在大多数情况下的阴影比一些距离在游戏中明显不会失去！

With no shadow cascades, hard shadows and shadow distance set to 20 units our shadows look like picture below. Note that shadows do fade out in the distance, but at the same time shadow quality is much better than it was with no cascades and a distance of 50 units. 由于没有影子级联，硬阴影和阴影距离设置为20个单位的阴影看起来像下面的图片。请注意，阴影不消失在远方，但同时阴影质量明显优于它没有级联和50个单位的距离。



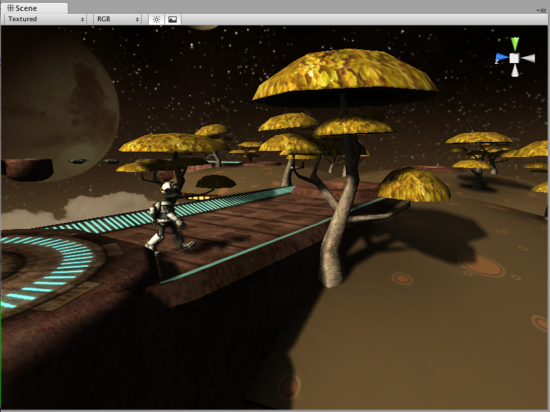
*Hard shadows with distance of 20 and no cascades.* 20的距离硬阴影而没有级联。

If on the other hand we set shadow distance too high, shadows won't look good at all. Setting distance to 100 here only decreases both performance and quality and does not make much sense - no objects in the scene are further than about 50 meters anyway! 如果在另一方面，我们设置阴影距离过高，阴影根本不会好看。这里设置距离只有100，既降低性能和质量而没有太大意义-在场景里无路如何没有对象进一步超过50米！



*Hard shadows with distance of 100 and no cascades. Ouch!* 100距离的硬阴影和没有级联。唉哟！

Shadow maps with cascades scale with distance much better. For example, four cascade soft shadows with covering 300 units in front of the camera look like picture below. It's somewhat worse than the picture at the top of this page, but not very bad either for a 6x increase in shadowing distance (of course in this scene that high shadow distance does not make much sense). 与级联的距离伸缩更好的阴影地图。例如，在覆盖300个单位的4级联柔阴影在摄像机前面看起来像下面的图像。这有点不如在此页顶部图片，但不是非常糟糕，在阴影的距离增加6倍（当然，在这个场景里，高阴影距离并没有多大意义）。



*Soft shadows with distance of 300 and four cascades.* 300距离的软阴影和4个级联

[**Unity Manual**](http://unity3d.com/support/documentation/Manual/index.html) > [**Advanced**](http://unity3d.com/support/documentation/Manual/Advanced.html) > [**Shadows in Unity**](http://unity3d.com/support/documentation/Manual/Shadows.html) > [**Troubleshooting Shadows**](http://unity3d.com/support/documentation/Manual/Shadow%20Troubleshooting.html)

Unity手册->高级->Unity中的阴影->阴影疑难解答

# Troubleshooting Shadows 阴影疑难解答

This page lists solutions to common [shadow](http://unity3d.com/support/documentation/Manual/Shadows.html) problems. 此页面列出了公共的阴影问题的解决办法。

### I see no shadows at all! 我根本看不到阴影

Shadows are a Unity Pro only feature. If you have Unity Indie, you won't get shadows. Simpler shadow methods, like using a [Projector](http://unity3d.com/support/documentation/Components/class-Projector.html), are still possible of course. 阴影是一个Unity专业版本唯一的功能。如果你是Unity的独立版本，你将不会得到阴影。简单的阴影的方法，像使用一个投影机，是可行的做法。

Shadows also require certain graphics hardware support. See [Shadows](http://unity3d.com/support/documentation/Manual/Shadows.html) page for details. 阴影还需要一定的图形硬件支持。详情请参阅阴影网页。

Check if shadows are not completely disabled in [Quality Settings](http://unity3d.com/support/documentation/Components/class-QualitySettings.html). 检验，在质量设置里，阴影不能设置为禁用。

### Some of my objects do not cast or receive shadows我的一些对象不能消除或接收阴影

First, the [Renderer](http://unity3d.com/support/documentation/Components/class-MeshRenderer.html) has to have Receive Shadows on to have shadows on itself; and Cast Shadows on to cast shadows on other objects (both are on by default). 首先，渲染器在它自己身上的阴影必须接收阴影，并使人对其它物体阴影投射阴影（两者都默认情况下）。

Next, only opaque objects cast and receive shadows; that means if you use built-in [Transparent](http://unity3d.com/support/documentation/Components/shader-TransparentFamily.html) or Particle shaders then you'll get no shadows. In most cases it's possible to [Transparent Cutout](http://unity3d.com/support/documentation/Components/shader-TransparentCutoutFamily.html) shaders (for objects like fences, vegetation etc.). If you use custom written [Shaders](http://unity3d.com/support/documentation/Manual/Shaders.html), they have to be pixel-lit and use [Geometry render queue](http://unity3d.com/support/documentation/Components/SL-SubshaderTags.html). Objects using **VertexLit** shaders do not receive shadows either (but can cast shadows just fine).

其次，只有不透明物体投下阴影和接收阴影，这意味着如果你使用内置的透明或粒子的着色器，然后你会得到没有阴影。在大多数情况下是可能的透明抠图着色（像围栏对象，植被等）。如果你使用常规的书面阴影，它们必须像素照明，并用几何渲染队列。对象使用VertexLit着色也不会收到任何阴影（但恰好可以投下阴影）。

Finally, only Pixel lights cast shadows. If you want to make sure that some light always casts shadows, no matter how many other lights are in the scene, set it to Force Pixel render mode (see [Light](http://unity3d.com/support/documentation/Components/class-Light.html) documentation).

最后，只有像素灯投射阴影。如果你想确保一些光源总是蒙上阴影，无论有多少其它光源在场景中的，将它设置为强制像素渲染模式（见光源文档）。

[**Unity Manual**](http://unity3d.com/support/documentation/Manual/index.html) > [**Advanced**](http://unity3d.com/support/documentation/Manual/Advanced.html) > [**Shadows in Unity**](http://unity3d.com/support/documentation/Manual/Shadows.html) > [**Shadow Size Computation**](http://unity3d.com/support/documentation/Manual/Shadow%20Size%20Details.html)

Unity手册->高级->Unity中的阴影->阴影大小计算

**Shadow Size Computation阴影大小计算**

Unity computes [shadow map](http://unity3d.com/support/documentation/Manual/Shadows.html) sizes this way: Unity计算阴影图像大小这种方法：

First light's "coverage box" on the screen is computed. This is what rectangle on the screen the light possibly illuminates: 在屏幕上的第一个灯光的“覆盖框”被计算。这是在屏幕上可能的光照矩形：

* For Directional lights that is the whole screen. 对于定向光源是整个屏幕。
* For Spot lights it's the bounding rectangle of light's pyramid projected on the screen. 对于聚光光源它是在屏幕上光源的金字塔规划的矩形边界。
* For Point lights it's the bounding rectangle of light's sphere projected on the screen. 对于点光源它是在屏幕上光源的球面规划的矩形边界。

Then the larger value of this box' width & height is chosen; call that pixel size. 然后，此框的较大宽度与高度的值选择，调用像素的大小。

At "High" shadow resolution, the size of the shadow map then is: 在“高”影子决定，阴影图像的大小则是：

* Directional lights: NextPowerOfTwo( pixel size \* 1.9 ), but no more than 2048.
* 定向光源：NextPowerOfTwo( pixel size \* 1.9 ),但是不能超过2048
* Spot lights: NextPowerOfTwo( pixel size ), but no more than 1024.
* 聚光光源：NextPowerOfTwo( pixel size )，但是不能超过1024
* Point lights: NextPowerOfTwo( pixel size \* 0.5 ), but no more than 512.
* 电光源：NextPowerOfTwo( pixel size \* 0.5 )，不能超过512

When graphics card has 512MB or more video memory, the upper shadow map limits are increased (4096 for Directional, 2048 for Spot, 1024 for Point lights).

当显卡拥有512M或更过视频内存，高阴影图片被限制增加（定向光源为4096、聚光光源是2048、点光源是1024）。

At "Medium" shadow resolution, shadow map size is 2X smaller than at "High" resolution. And at "Low" resolution, it's 4X smaller than at "High" resolution. 在中等阴影决定，阴影图像的大小是2倍小于高决定。而在“低”的决定，它是4倍小于高决议。

The seemingly low limit on Point lights is because they use cubemaps for shadows. That means six cubemap faces at this resolution must be in video memory. They are also quite expensive to render, as potential shadow casters must be rendered into up to six cubemap faces. 在点光源上最小尺寸看起来似乎是因为他们使用的立方体图片的阴影。这意味着6个立方体面临着在这项决定必须在视频内存。它们也相当昂贵的渲染，作为潜在的阴影脚轮必须渲染成6个立方体的表面。

**Shadow size computation when running close to memory limits 阴影大小计算接近运行时内存限制**

When running close to video memory limits, Unity will automatically drop shadow map resolution computed above. 当运行接近视频内存的限制，Unity将自动下落阴影地图上述决议计算。

Generally memory for the screen (backbuffer, frontbuffer, depth buffer) has to be in video memory; and memory for render textures has to be in video memory, Unity will use both to determine allowed memory usage of shadow maps. When allocating a shadow map according to size computed above, it's size will be reduced until it fits into (TotalVideoMemory - ScreenMemory - RenderTextureMemory) / 3.

概括而言，屏幕内存（后缓冲，前缓冲，深度缓冲）必须在视频内存里，必须在视频内存提供纹理内存，Unity将同时使用，以确定允许内存使用的阴影地图。当分配一个阴影图根据以上的规模计算，它的大小将减少直到它（TotalVideoMemory - ScreenMemory - RenderTextureMemory）/ 3相符。

Assuming all regular textures, vertex data and other graphics objects can be swapped out of video memory, maximum VRAM that could be used by a shadow map would be (TotalVideoMemory-ScreenMemory-RenderTextureMemory). But exact amounts of memory taken by screen and render textures can never be determined, and some objects can not be swapped out, and performance would be horrible if all textures would be constantly swapping in and out. So Unity does not allow a shadow map to exceed one third of "generally available" video memory, which works quite well in practice.

假设所有正规的纹理，顶点数据和其他图形对象可以被交换出去的视频内存，最大显存可以通过地图使用的阴影将是（TotalVideoMemory - ScreenMemory - RenderTextureMemory）。但是，具体数额由屏幕所呈现的纹理内存永远不能确定，有些对象不能被交换出去，如果所有的纹理会不断交换和退出，性能将是可怕的。因此，Unity不允许阴影地图超过三分之一的“普遍”的视频内存，其运作方式在实践中相当不错。

[**Unity Manual**](http://unity3d.com/support/documentation/Manual/index.html) > [**Advanced**](http://unity3d.com/support/documentation/Manual/Advanced.html) > [**Optimizing Graphics Performance**](http://unity3d.com/support/documentation/Manual/Optimizing%20Graphics%20Performance.html)

Unity手册->高级->优化图形性能

**Optimizing Graphics Performance 优化图形性能**

Making your game run smoothly is of prime importance to its success. Thankfully Unity is there for you! We have spent a lot of time and energy making it run fast on a wide variety of hardware. Below are some simple guidelines to maximizing the speed of your game.

使你的游戏顺利运行是最重要的成功。值得庆幸的Unity为你准备了！我们花了很多时间和精力使它在多种硬件快速运行。以下是最大限度地发挥你的游戏速度的一些简单的指导方针。

**In summary - combine, combine, combine 总的来说-组合，组合，组合**

* If you care about performance, combine meshes. 如果你关心性能，结合网格。
* If you care about performance, make sure all your combined meshes also share the same material and texture. 如果你关心性能，请确保你所有的组合网格也有同样的材料和纹理。
* [Profiler](http://unity3d.com/support/documentation/Manual/Profiler.html) and [Rendering Statistics](http://unity3d.com/support/documentation/Manual/RenderingStatistics.html) window are very helpful!
* 分析器和渲染统计窗口是非常有帮助！

**In detail: 详细**

Modern graphics cards are really good at pushing a lot of polygons, but they have quite a bit of overhead for every batch that you submit to the graphics card. So if you have a 100-triangle object it is going to be just as expensive to render as a 1500-triangle object. The sweet spot for optimal rendering performance is somewhere around 1500-4000 triangles per mesh.

现代显卡擅长推动很多的多边形，但他们仍有很多开销为每一批你提交的显卡。因此，如果你有一个100三角形对象它将是一样昂贵去作为1500三角形对象渲染。达到最佳的渲染性能是每网格1500-4000三角形。

You only pay a rendering cost for objects that have a **Mesh Renderer** attached. And you only pay for those that are within the view frustum. There is no rendering cost from having a lot of empty **GameObjects** in your scene.

你只需为对象给予一个网格渲染器附加的一个渲染。你只需给予对那些视域范围内的。目前没有渲染的费用为许多空白GameObjects在你的场景里。

* The best way to improve rendering performance is to combine objects together so each mesh has around 1500 or more triangles and uses only one **Material** for the entire mesh.
* 最好的办法提高渲染性能是结合在一起的对象，因此每个网格拥有大约1500或更多的三角形，并使用只有一个材质为整个网格。

It is important to understand that just combining two objects which don't share a material does not give you any performance increase at all. If you want to combine effectively, you need to make sure your mesh uses only one material after you have combined it.

重要要明白，只有两个对象的组合将不会给你根本任何性能的提升。如果你想有效结合，你需要确保你的网格只使用一个的材料在你组合它以后。

There is one thing to be aware of when combining objects though: if you use a lot of small lights in your scene, it might make sense to combine only objects that are close to each other.

有一件事要知道当组合对象时：如果在你的场景里使用许多小的光源，它或许是明智的结合相互靠近的对象。

The rendering cost for a mesh that has multiple materials is the same as having multiple renderers for each material. The most common reason why you have multiple materials is because two meshes don't share the same textures. So, if you want to optimize rendering performance, you need to make sure that the objects you combine share textures.

在一个具有多个网格材料的渲染成本是每一个的材料具有多个相同的渲染。最常见原因有多种材料因为这两个网格不共享相同的纹理。所以，如果你要优化渲染性能，你需要确保你组合的对象共享纹理。

* Unity is very good at pushing lots of polygons. Unity uploads all geometry to the graphics card for good cache utilization and optimal data alignment.
* Unity是非常善于推动大量的多边形。Unity上传到所有的几何给适合缓存的显卡应用和和优化数据。
* You simply have to make sure that the graphics card doesn't have to handle large numbers of batches.
* 你只需确保显卡不要处理大量的批次。
* The number of **Pixel Lights** affecting an object heavily affects performance.
* 光源的像素数量严重影响对象性能。

If you want to have the best performance and don't care about Bumpmapping or Pixel Lighting, go to **Edit->Render Settings...** and set **Pixel Light Count** to zero. This will simply use vertex lighting for all objects. This means all geometry will be rendered only once per frame. This is an extremely useful LOD setting, so your game can run fine on older graphics cards.

如果您想拥有最好的性能而不关心凸起块映射或像素光源，进入**Edit->Render Settings...**并设置像素光源为零。这将对所有对象简单的使用顶点光源。这意味着所有几何将每帧渲染仅一次。这是一个非常有用的LOD设置，使你的游戏可以良好的运行在老的显卡上。

**Pixel lights 像素光源**

If you use pixel lighting, then each GameObject has to be rendered as many times as there are pixel lights that affect the object. If you combine two objects that are very far apart, it might increase the size of the object and now you have a lot of lights affecting this big object. If your objects were separate however, the light wouldn't have to be applied on the part of the mesh which is far away. This can result in rendering the combined mesh as many times as the uncombined mesh thus you didn't save anything. For this reason, you should keep GameObjects that are very far away as individual Meshes.

如果你使用像素光源，那么每个GameObject要尽可能多的时间有响的对象的像素光源被渲染。如果你组合的两个对象是非常遥远，它可能会增加对象的大小而且现在你有许多光源影响这个大的对象。然而如果你的对象是分开的，光源将不必对遥远网格的一部分应用。这可能导致渲染的未结合的网格多次联合网格，因此你有保存任何东西。为此，你应该保持GameObjects是很遥远的个人网格。

When rendering a mesh, Unity finds all lights surrounding the mesh. It then figures out what lights affect the mesh the most. The **Edit->Render Settings** are used to modify how many of the lights end up as pixel lights and how many as vertex lights.

当渲染网格，Unity发现所有的光源周围的网格。然后它估算出是什么光源影响网格最大。**Edit->Render Settings**用来修改多少光源作为像素光源结束和多少光源作为顶点光源结束。

Every light calculates its importance based on how far away it is from the mesh and how intense it is.

每一个光源计算它自己的重要性基于它是距离网格多远和它是多么激烈的（光源亮度的激烈程度）。

Some lights are more important than others depending on the game context. For this reason, every light has a **Render Mode** setting which can be set to **Force Pixel** or **Force Vertex**.

有些光源重要性超过其它光源，依据游戏的上下文。因为这个理由，每一个光源有一个染模式设置，可被设置强制像素光源或强制顶点光源。

Imagine the player's car with head lights driving through the night. The head light is the most important light in the game. For this reason, the head lights Render Mode should be set to **Force Pixel**.

设想玩家通过夜间驾驶的汽车的头灯光。头灯光是游戏中最重要的光。因为这个理由，头灯光渲染模式应设置为强制像素。

If you have a light that isn't very important and also visually doesn't gain much from being a pixel light, set the lights Render Mode to "Force Vertex". This way you don't waste rendering performance or lose any visual quality.

如果你有一个不是很重要光并且也不能获得视觉上从一个像素的很多光线，设置灯光渲染模式为强制顶点。这样，你就不会浪费的渲染性能或损失任何视觉质量。

**Per-layer cull distances每一层精选距离**

You might want to cull small objects earlier to reduce number of draw calls. For example, small rocks and debris could be made invisible at much smaller distance than large buildings. To do that, put small objects into a [separate layer](http://unity3d.com/support/documentation/Components/Layers.html) and setup per-layer cull distances using [Camera.layerCullDistances](http://unity3d.com/support/documentation/ScriptReference/Camera-layerCullDistances.html) script function.

你可能想精选早期小对象减少绘制调用的数量。例如，小石头和瓦砾应能在更小的距离超过无形的大型建筑物。要做到这一点，放入一个单独的层和设置每层精选距离使用Camera.layerCullDistances脚本函数。

**Shadows 阴影**

Shadows are generally expensive. They can add a lot of performance overhead to your game if they are not used correctly. For more details about shadows, please read the [Shadows page](http://unity3d.com/support/documentation/Manual/Shadows.html).

阴影一般昂贵。如果没有正确使用阴影，它们可以添加大量的性能开销到以你的游戏。有关阴影详细信息，请阅读阴影页面。

**See Also 参见**

* [Modeling Optimized Characters](http://unity3d.com/support/documentation/Manual/Modeling%20Optimized%20Characters.html) 模型优化角色
* [Optimizing for integrated graphics cards](http://unity3d.com/support/documentation/Manual/OptimizeForIntegratedCards.html) 集成显卡优化
* [Rendering Statistics Window](http://unity3d.com/support/documentation/Manual/RenderingStatistics.html) 渲染统计窗口

[**Unity Manual**](http://unity3d.com/support/documentation/Manual/index.html) > [**Advanced**](http://unity3d.com/support/documentation/Manual/Advanced.html) > [**Optimizing Graphics Performance**](http://unity3d.com/support/documentation/Manual/Optimizing%20Graphics%20Performance.html) > [**Modeling Optimized Characters**](http://unity3d.com/support/documentation/Manual/Modeling%20Optimized%20Characters.html)

Unity手册->高级->优化图形性能->模型优化角色

# Modeling Optimized Characters 模型优化角色

## Use one Skinned Mesh Renderer 使用一个皮肤网格渲染器

Your character should use only a single [Skinned Mesh Renderer](http://unity3d.com/support/documentation/Components/class-SkinnedMeshRenderer.html). There is usually no reason to use multiple meshes for a character. Unity also has optimizations related to visibility culling and bounding volume updating which only kick in if you use one [animation component](http://unity3d.com/support/documentation/Components/class-Animation.html) and one skinned mesh renderer in conjunction. If you care about performance, multiple skinned meshes per character is **not** an option. If you use two skinned mesh renderers for one character instead of one, the time spent on rendering the character will most likely double!

你的角色应该只使用一个单一的皮肤网格渲染。通常没有理由使用一个多网格为一个角色。Unity也有与精选可见和卷更新结合的优化，仅交付如果你使用一个动画组件和一个皮肤网格渲染器在组合里。如果你关心性能，每个角色的多种网格皮肤不是一种选择。如果你为一个角色使用两个网格渲染器替代一个，在渲染角色上花费的时间将是双倍！

## Don't use many Materials 不要使用许多材质

You also want to keep the number of [Materials](http://unity3d.com/support/documentation/Components/class-Material.html) on that Mesh as low as possible. There is only one reason why you might want to have more than one material on the character: when you need to use a different shader (e.g. if you want to use a special shader for the eyes). However 2-3 Materials per character should be sufficient in almost all cases. If your character is carrying a gun, it might be useful to have the gun a separate object, simply because it might get detached.

你也希望能够保持尽可能低的网格材料的数量。只有一个原因，你可能想有一个以上的材料的在角色上：当你需要使用不同的阴影（例如，如果你要使用一个特殊的眼睛着色）。但是每个角色的2-3材料应足以满足几乎所有情况。如果你的角色是带着枪，可能通常用于枪的一个单独的对象，只是因为它可能会脱落。

## Reduce amount of bones 减少骨骼的数量

Medium games use bone hierarchies with 15-60 bones. The fewer bones you use the faster it will run and with 30 bones you can do fairly good quality. Unless you really have to, we strongly recommend you use around 30 bones per character.

中等游戏使用15-60骨层次结构。你使用的少量的骨它将运行的快而利用30个骨你可以做到较好的质量。除非你真的要，我们强烈建议你大约30个角色骨头的使用。

## Polygon count 多边形计数

How many polygons you should use depends on the quality you are going after. Anything between 500-6000 triangles is reasonable. If you want lots of characters on screen, you will have to sacrifice in polycount per character, if you want it to run on old machines, you will have to use less polygons per character. As an example: Half Life 2 characters used 2500-5000 triangles per character. Next-gen AAA games running on PS3 or Xbox 360 usually have characters with 5000-7000 triangles.

你应该使用多少多边形取决于你是追求的质量。任何事情在500-6000之间的三角形是合理的。如果你想在屏幕上很多角色，你将不得不牺牲每个角色的策略数，如果你想让它老机器上运行，你将不得不每个角色使用少多边形。作为一个例子：半条命2个角色每个角色使用2500-5000三角形。下一代AAA级游戏运行在PS3或Xbox 360上通常有5000-7000三角形的角色。

## Separate out IK and FK 隔开IK和FK

Separate out Inverse Kinematics (IK) and Forward Kinematics (FK). When animations are imported the IK nodes are baked into FK, thus Unity doesn't need the IK nodes at all. You can either kill the **GameObjects** in Unity or the nodes in the modelling tool. By removing them, the IK nodes don't need to be animated every frame anymore. For this reason it is a very bad idea to intermix IK and FK hierarchies. Instead you should create two hierarchies: one strictly for IK and one for FK. This way you can very easily select the whole IK hierarchy and delete it.

隔离出逆向运动（IK）和正向运动（FK）。当动画导入的节点被烘烤到FK里，然而unity根本不需要IK节点。你也可以删除在Unity里的GameObjects或在建模工具里的节点。通过删除它们，IK节点不需要每帧的动画了。为此，它是一个非常糟糕的主意混杂IK和FK层次结构。相反，你应该创建两个层次：一是绝对的IK层和一个绝对的FK层。这种方式可以很容易地选择整个IK层并将其删除。

## Use reusable rigs 重复利用装备

Create a rig which you can reuse. This allows you to share animations between different characters.

创建一个你可以重复使用装备。这允许你可以共享动画不同的角色。

## Name bones correctly 正确命名骨骼

Name bones correctly (left hip, left ankle, left foot etc.). Especially with characters, naming your bones correctly is very important. For example if you want to turn a character into a Ragdoll you will have to find the right bones for each body part. If they are not named correctly, finding all the body parts will take much longer.

正确命名骨骼（左臀部，左脚脚踝，左脚等）。特别是与人物，正确地命名你的骨骼是非常重要的。例如，如果你想一个角色变成一个布娃娃你必须找到每个部位的正确骨骼。如果它们不正确命名，发现身体的所有部位将需要更长的时间。

[**Unity Manual**](http://unity3d.com/support/documentation/Manual/index.html)>[**Advanced**](http://unity3d.com/support/documentation/Manual/Advanced.html)> [**Optimizing Graphics Performance**](http://unity3d.com/support/documentation/Manual/Optimizing%20Graphics%20Performance.html) > [**Optimizing for integrated graphics cards**](http://unity3d.com/support/documentation/Manual/OptimizeForIntegratedCards.html)

Unity手册->高级->优化图形性能->优化集成显卡

**Optimizing for integrated graphics cards**优化集成显卡

**Polygon count matters 多边形计算材料**

On most graphics cards today, polygon count does not really matter. The common knowledge is that object count and fillrate is much more important. Unfortunately, not so on the majority of integrated chips (most of them being Intel cards). Every polygon matters. How much it matters depends on the complexity of the vertex shaders or lighting and the speed of the CPU (thats right, most integrated cards transform & light vertices on the CPU).

在大多数显卡今天，多边形数量其实并不重要。对象数量和填充率的常识是非常重要的。不幸的是，不是对多数（其中大多数是英特尔卡）集成芯片。每个多边形内容。多少取决于它的内如对顶点着色或照明设备和CPU速度的复杂性。（集成度最高的显卡变换和顶点光在CPU上）

[Big Bang Brain Games](http://www.freeverse.com/braingames/) never went above 25,000 triangles in a scene using 1-2 vertex lights and no pixel lights. Quality Settings were used to speed up performance automatically when frame rate drops. So on higher end machines a higher quality setting was used which had pixel lights enabled.

大爆炸电脑游戏从来没有超过25,000三角形在一个场景里使用1-2顶点光源和没有像素光源。质量设置被用来自动加速性能当帧速率下降时。所以在高端更高质量的设置是用启用高精度的像素光源。

What slows things down is drawing objects multiple times, using complex vertex shaders and lots of polygons. This means:

什么东西慢下来是绘图对象多次，使用复杂的顶点着色器和大量的多边形。这意味着：

* Do not use pixel lights. Pixel lights make the object be drawn multiple times; once for ambient pass and once for each pixel light.
* 不要使用像素光源。像素光源使对象绘制多次，一次环绕通过和一次每个像素光源。
* Try not to use lights at all, even vertex lights. Lights make sense if your geometry moves, or if your lights move. Otherwise bake the illumination, it will run faster and look better.
* 不尝试使用所有的光源，甚至顶点光源。光源有意义，如果你的几何移动，或者如果你的光源移动。否则烤照明度，它将运行快速以及看起来更好。
* Optimize your geometry (see section below). 优化你的几何（见下节）。
* Use [Rendering Statistics](http://unity3d.com/support/documentation/Manual/RenderingStatistics.html) window and [Profiler](http://unity3d.com/support/documentation/Manual/Profiler.html)! 使用渲染统计窗口和分析器！

**Optimize model geometry 优化模型几何结构**

When optimizing the geometry of a model, there are two basic rules:

当优化一个模型的几何结构，有2个基本规则：

* Don't use excessive amount of faces if you don't have to. 不要使用过多的表面，如果你不是必须的。
* Keep the number of UV mapping seams and hard edges as low as possible.
* 保持UV图像接缝和尽可能低的硬边毛刺。

Note that the actual number of vertices that graphics hardware has to process is usually not the same as displayed in a 3D application. Modeling applications usually display the geometric vertex count, i.e. number of points that make up a model.

请注意，顶点的实际数目图形硬件的过程通常是不一样的在3D应用里显示出来。建模应用程序通常显示几何顶点计数，例如：点的数即构成一种模式。

For a graphics card however, some vertices have to be split into separate ones. If a vertex has multiple normals (it's on a "hard edge"), or has multiple UV coordinates, or multiple vertex colors, it has to be split. So the vertex count you see in Unity is almost always different from the one displayed in 3D application.

然而对于显卡，一些顶点必须纳入单独的分开。如果一个顶点有多个标准（这是一个“硬优势”），或有多个UV坐标，或多个顶点颜色，它必须被分割。因此，你看到顶点计数在Unity里几乎总是不同于在3D应用程序里的一个显示。

**Bake lighting. 烤光源**

Bake ligthing either into lightmaps or vertex colors.烤光源要么进入到光映射或顶点颜色。

Maya has good builtin lightmapping tools. Unity can import secondary UV maps or vertex colors from Maya.

Maya具有良好的内建光源映射工具。Unity可以从Maya导入紫外线地图或顶点颜色。

The process of generating a lightmapped environment takes a little longer than just placing a light in the scene in Unity, **but**:

在生成处理一个光源映射环境需要一点长的时间仅是放置一个光源在Unity的场景里，但是：

* it's going to run a lot faster (2-3 times for eg. 2 lights). 这将运行快得多（2-3倍对于2个光源）。
* and look a lot better since you can bake global illumination and the ligthmapper can smooth the results. 并期待好的多，因为你可以烤全局照明度和光映射可以平滑的结果。

Even next-gen games like Gears of War still rely on lightmapping a lot. Usually they use lightmapped environments and place dynamic lights on top. The builtin Lightmapped shaders can do exactly that in Unity!

即使未来像战争机器游戏仍然依赖于光映射很多。通常它们使用光源映射环境和放置动态光在上面.内置的光映射阴影可以做到这点在Unity里！

[**Unity Manual**](http://unity3d.com/support/documentation/Manual/index.html) > [**Advanced**](http://unity3d.com/support/documentation/Manual/Advanced.html) > [**Optimizing Graphics Performance**](http://unity3d.com/support/documentation/Manual/Optimizing%20Graphics%20Performance.html) > [**Rendering Statistics Window**](http://unity3d.com/support/documentation/Manual/RenderingStatistics.html)

Unity手册->高级->优化图形性能->渲染统计窗口

**Rendering Statistics Window 渲染统计窗口**

The **Game View** has a **Stats** button top right. When this Stats button is pressed, an overlay window is displayed with realtime rendering statistics. This is very useful for [Optimizing Graphics Performance](http://unity3d.com/support/documentation/Manual/Optimizing%20Graphics%20Performance.html). Additionally, [Profiler](http://unity3d.com/support/documentation/Manual/Profiler.html) shows some rendering statistics.

游戏视图有一个统计按钮在右上角。当这个按钮被按下统计，一个重叠窗口与实时渲染的统计数字显示。这是非常有用的优化的图形性能。此外，分析器显示了一些渲染的统计数字。



*Rendering Statistics Window* 渲染统计窗口

Statistics window contains the following information: 通体窗口包含如下的信息：

|  |  |
| --- | --- |
| **Time per frame and FPS**  **每帧时间和FPS** | How much time it takes to process and render one game frame (and resulting FPS). Note that this number only includes frame update and rendering of the game view; and does not include time taken in the editor to draw the scene view, inspector and other editor-only processing.  花费多少的时间去处理进程和渲染一个游戏的帧（以及FPS结果）。请注意，这个数字只包括帧更新和游戏图像生成;，不包括在编辑器中绘制场景视图，检视器和其它编辑器只处理时间。 |
| **Draw Calls**  **绘制调用** | How many objects are drawn in total. This accumulates objects that are drawn multiple times as well, for example some object that is affected by pixel lights will add several draw calls.  总共有多少对象被绘制。这种积累的对象也被绘制多次，例如一些受影响对象通过像素光源绘制将添加几个绘制调用。 |
| **Tris** and **Verts**  **三角形和顶点** | Number of triangles and vertices drawn. This is mostly important when [optimizing for low-end hardware](http://unity3d.com/support/documentation/Manual/OptimizeForIntegratedCards.html)  三角形和顶点数目绘制。这主要是重要的当优化低端硬件时。 |
| **Used Textures**  **使用纹理** | Count and memory size of textures used when drawing this frame.  纹理的数量和内存大小使用当绘制这个帧时。 |
| **Render Textures**  **渲染纹理** | Count and memory size of [Render Textures](http://unity3d.com/support/documentation/Components/class-RenderTexture.html) that are created. Also displays how many times active Render Texture was switched during this frame.  所创建的纹理渲染的数量和内存大小。也显示有多少次激活的渲染纹理被在这个帧期间交换。 |
| **Screen**  **屏幕** | Size, anti-aliasing level and memory taken by the screen itself.  大小，图形保真水平和内存携带通过屏幕它自身。 |
| **VRAM usage**  **显存使用率** | Approximate bounds of current video memory (VRAM) usage. Also shows how much video memory your graphics card has.  当前视频内存（显存）使用近似边界。也显示你显卡有多少视频内存。 |
| **VBO total**  **VBO总计** | Number of unique meshes (vertex buffers) that are uploaded to the graphics card. Each different model will cause new VBO to be created. In some cases scaled objects will cause additional VBOs to be created.  独特网格的数量（顶点缓冲器）上传至显卡。每一个不同的模型将导致新的VBO被创建。在某些情况下伸缩对象将造成额外VBOS被创建。 |
| **VB uploads**  **VB 上载** | Count and size of vertex data changed this frame. Dynamic geometry, like particles or skinned meshes, will contribute to this, as well as terrain LOD changing, etc.  这个帧的顶点数据更改的数量和大小。动态几何结构、像粒子和皮肤网格，将有助于这一点，以及地形LOD的变化等。 |
| **IB uploads**  **IB 上载** | Count and size of triangle data changed this frame. Dynamic geometry, like particles or skinned meshes, will contribute to this, as well as terrain LOD changing, etc.  这个帧的三角形数据更改的数量和大小。动态几何结构，像粒子和皮肤网格，将有助于这一点，以及地形LOD的变化等。 |
| **Visible Skinned Meshes**  可视皮肤网格 | How many skinned meshes are rendered.  多少皮肤网格被渲染。 |
| **Animations**  **动画** | How many animations are playing.  多少动画 被播放。 |

[**Unity Manual**](http://unity3d.com/support/documentation/Manual/index.html) > [**Advanced**](http://unity3d.com/support/documentation/Manual/Advanced.html) > [**Reducing File Size**](http://unity3d.com/support/documentation/Manual/Reducing%20File%20size.html)

Unity手册->高级->减少文件大小

**Reducing File Size 较少文件大小**

**Unity post-processes all imported assets**

**Unity后处理所有导入的资产**

Unity always post-processes imported files, thus storing a file as a multi-layered psd file instead of a jpg will make absolutely zero difference in the size of the player you will deploy. Save your files in the format you are working with (eg. .mb files, .psd files, .tiff files) to make your life easier.

Unity始终后处理导入的文件，因此存储一个文件作为一个多层次的psd文件代替一个jpg文件将绝对零差别在你部署的播放器大小里。保存你文件在正在使用（例如.MB 、.PSD.、TIFF格式文件）格式使你的生活更轻松。

**Unity strips out unused assets Unity输出不使用的资产**

The amount of assets in your project folder does **not** influence the size of your built player. Unity is very smart about detecting which assets are used in your game and which are not. Unity follows all references to assets before building a game and generates a list of assets that need to be included in the game. Thus you can safely keep unused assets in your project folder.

在你的项目文件夹里的资产的数量不影响你的内置播放器的大小。Unity是很聪明关于检测哪些资产应用与你的游戏中，哪些不是。Unity跟踪所所有的引用资产在建立一个游戏之前并产生一个需要包含到游戏里的资产列表。因此，你可以安全地保持没有使用的资产在你的项目文件夹里。

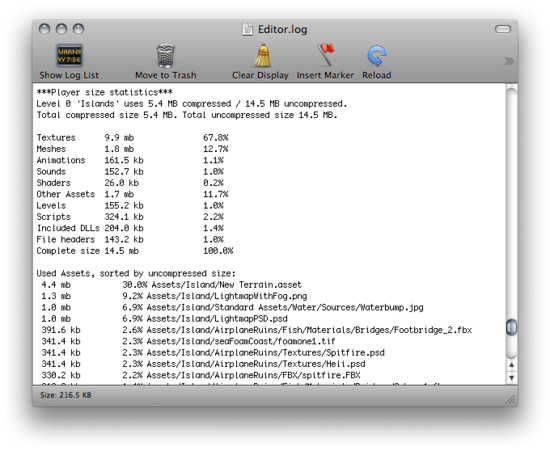
**Unity prints an overview of the used file size Unity打印出使用文件大小的概述**

After Unity has completed building a player, it prints an overview of what type of asset took up the most file size, and it prints which assets were included in the build.

Unity已完成建设一个播放器后，它打印一个什么样的资产类型概述携带大多数的文件大小，以及它打印的包含在创建里的资产。

To see it just open the editor console log: **Open Editor Log** button in the Console window (**Window -> Console**).

去查看它只需打开编辑器控制日志：打开编辑器日志按钮在控制台窗口里（**Window -> Console**）。



*An overview of what took up space* 一个携带什么空间的概述

**Optimizing texture size 优化纹理大小**

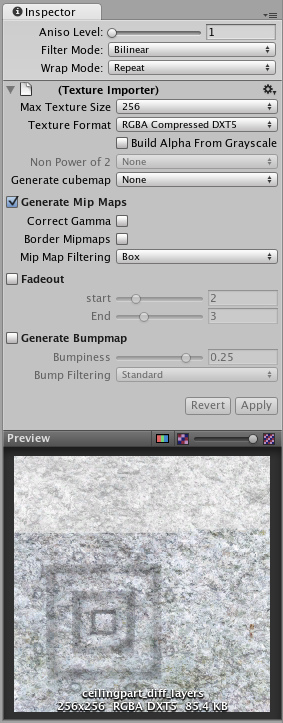
Often textures take up most space in the build. The first to do is to use compressed texture formats (DXT or PVRTC) where you can.

通常纹理携带许多空间在创建里。第一个做的是你可以在那里使用压缩纹理格式（DXT或PVRTC）。

If that doesn't get the size down, try to reduce the size of the textures. The trick here is that you don't need to modfiy the actual source content. Simply select the texture in the Project view and set **Max Texture Size** in Import Settings. It is a good idea to zoom in on an object that uses the texture, then adjust the **Max Texture Size** until it starts looking worse in the **Scene View**.

如果没有得到减少的大小，尝试减少纹理的大小。这里的技巧是，你不必修改实际来源的内容。简单地选择在项目视图的纹理和在导入设置里设置最大纹理尺寸。这是一个好主意放大在一个使用纹理的对象上，然后调整最大纹理尺寸直到它开始看起来不好看在场景视图里。

*Changing the Maximum Texture Size will not affect your texture asset, just its resolution in the game*



*更改纹理大小的最大值将不影响你的纹理资产，仅是它自己的分辨率在游戏里。*

**How much memory does my texture take up? 我的纹理占去多少内存**

|  |  |
| --- | --- |
| **Compression 压缩** | **Memory consumption 内存消耗** |
| RGB Compressed DXT1 | 0.5 bpp (bytes/pixel) |
| RGBA Compressed DXT5 | 1 bpp |
| RGB 16bit | 2 bpp |
| RGB 24bit | 3 bpp |
| Alpha 8bit | 1 bpp |
| RGBA 16bit | 2 bpp |
| RGBA 32bit | 4 bpp |

To figure out total texture size: width \* height \* bpp. Add 33% if you have Mipmaps.

计算出纹理大小总计：宽\*高\*bpp曾记33% 如果你有Mipmaps。

By default Unity compressed all textures when importing. This can be turned off in the **Preferences** for faster workflow. But when building a game, all not-yet-compressed textures will be compressed.

到导入时，Unity默认压缩所有的纹理。这可以在参数选择里关闭为加快工作流程。但当编译一个游戏时，所有没有压缩的纹理将被压缩。

**Optimizing mesh and animation size 优化网格和动画大小**

[Meshes](http://unity3d.com/support/documentation/Components/class-Mesh.html) and imported Animation Clips can be compressed so they take up less space in your game file. Compression can be turned on in Mesh Import Settings.

网格和导入的动画剪辑可以被压缩因此它们占去少的空间在你的游戏文件里。压缩可以开启在网格导入设置里。

Mesh and Animation compression uses quantization, which means it takes less space but the compression can introduce some inaccuracies. Experiment with what level of compression is still acceptable for your models.

网格和动画压缩使用量化，这意味着它占用少的空间但压缩可能引入一些不准确度。为你的模型实验什么样的压缩级别是可以接受的。

Note that mesh compression only produces smaller data files, and does not use less memory at run time. Animation **Keyframe reduction** produces smaller data files *and* uses less memory at run time, and generally you should always use keyframe reduction.

请注意，网格压缩只处理小型数据文件，并不在运行时使用较少的内存。动画关键帧还原生成较小的数据文件和在运行时使用较少的内存，通常你应该总是使关键帧减少。

Additionally, you can choose not to store normals and/or tangents in your Meshes, to save space both in the game builds and memory at run time. This can be set in **Tangent Space Generation** drop down in Mesh Import Settings. Rules of thumb:

此外，你可以选择不要在你的网格平均和/或切线，既节省空间，在游戏中建立和内存在运行时。这可以在导入设置里里切线空间生成里设置下降。经验法则：

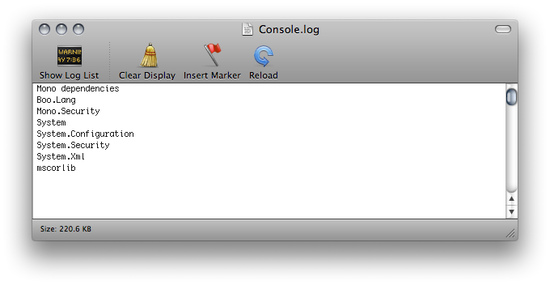
* Tangents are used for bump-mapping. If you don't use bump-mapping, you probably don't need to store tangents in those meshes.
* 切线用于凹凸映射。如果你不使用凹凸映射，你可能并不需要存储在这些网格切线。
* Normals are used for lighting. If you don't use realtime lighting on some of your meshes, you probably don't need to store normals in them.
* 正常用于照明。如果你不使用你的网格一些实时照明，你可能不需要存储正常在它们里。

**Reducing included dlls in the Web Player 减少包含的dlls在web播放器里**

When building a Web player it is important to not depend on **System.dll** or **System.Xml.dll**. Unity does not include **System.dll** or **System.Xml.dll** in the web player installation. That means, if you want to use Xml or some Generic containers which live in **System.dll** then the required dlls will be included in the web player. This usually adds 1mb to the download size, obviously this is not very good for the distribution of your web player and you should really avoid it. If you need to parse some Xml files, you can use a smaller xml library like this one [Mono.Xml.zip](http://www.unity3d.com/examples/Mono.Xml.zip). While most Generic containers are contained in mscorlib, Stack<> and few others are in **System.dll**. So you really want to avoid those.

当建立一个Web播放器，重要的是不依赖于System.dll中或System.Xml.dll。Unity不包括在网页播放器安装System.dll或System.Xml.dll。这意味着，如果你想使用XML或某些通用容器在System.dll里那么所需要的DLL将包含在WEB播放器中。这通常增加下载1MB的大小，显然这不是很好的对你的WEB播放器，你应该避免。如果你需要分析一些XML文件，你可以使用类似这样的Mono.Xml.zip一个较小的XML库。虽然大多数通用容器包含mscorlib，Stack和其他几个都是在System.dll里。因此你真正要避开这些。

*As you can see, Unity is including System.Xml.dll and System.dll, when building a player*



*正如你看到的，Unity是包含System.Xml.dll 和 System.dll，当编译一个播放器时。*

Unity includes the following DLLs with the web player distribution **mscorlib.dll**, **Boo.Lang.dll**, **UnityScript.Lang.dll** and **UnityEngine.dll**.

Unity包含下面的DLLS利用web播放器发布mscorlib.dll、Boo.Lang.dll、nityScript.Lang.dll和UnityEngine.dll。

[**Unity Manual**](http://unity3d.com/support/documentation/Manual/index.html) > [**Advanced**](http://unity3d.com/support/documentation/Manual/Advanced.html) > [**Web Player Streaming**](http://unity3d.com/support/documentation/Manual/Web%20Player%20Streaming.html)

Unity手册->高级->web播放器流

**Web Player Streaming Web播放器流**

Web Player Streaming is critical for providing a great web gaming experience for the end user. The idea behind web games is that the user can view your content almost immediately and start playing the game as soon as possible instead of making him wait for a progress bar. This is very achievable, and we will explain how.

Web播放流是至关重要的提供良好的WEB游戏体验为最终用户。WEB游戏背后的想法是用户可以查看你的内容几乎尽快立即开始播放，而不是让他等待一个进度栏的游戏。这是完全实现的，我们将解释如何。

**Tuning for Portals 调整门户**

This section will focus on publishing to online game portals. Streaming is useful for all kinds of contents, and it can easily be applied to many other situations.

本节将集中发布到在线游戏门户网站。流媒体是有益的各种内容，并且可以很容易地应用到许多其他情况。

Online game portals expect that some form of game play really starts after downloading at most 1 MB of data. If you don't reach this makes it that much less likely for a portal to accept your content. From the user's perspective, the game needs to start quickly. Otherwise his time is being wasted and he might as well close the window.

在线游戏门户网站预计部分游戏的形式真正追求最多1 MB的数据下载启动。如果你没有达到这使得它不可能为一个门户网站接受你的内容。从用户的角度来看，游戏需要快速启动。否则，浪费他的时间，他也学关闭该窗口。

On a 128 kilobit cable connection you can download 16 KB per second or 1 MB per minute. This is the low end of bandwidth online portals target.

在一个 128千比特电缆连接，您可以下载16kB的每秒或每分钟1 MB。这是在线目标门户网站的带宽的低端。

The game would optimally be set up to stream something like this:

游戏将设立最佳流像这样：

1. 50 KB display the logo and menu (4 seconds) 50KB显示标识和菜单（4秒）
2. 320 KB let the user play a tiny tutorial level or let him do some fun interaction in the menu (20 seconds) 320 KB让用户发挥小教程水平或让他做一些有趣的互动在菜单条里（20秒）
3. 800 KB let the user play the first small level (50 seconds) 800 KB让用户打第一个小级别（50秒）
4. Finish downloading the entire game within 1-5 MB (1-5 minutes) 完成在1-5 MB的完整游戏下载（1-5分钟）

The key point to keep in mind is to think in wait times for a user on a slow connection. Never let him wait.

关键的一点要记住，想一想通过慢速连接用户是在等待时间。从来没有让他等待。

Now, don't panic if your web player currently is 10 MB. It seems daunting to optimize it, but it turns out that with a little effort it is usually quite easy to structure your game in this fashion. Think of each above step as an individual scene. If you've made the game, you've done the hard part already. Structuring some scenes around this loading concept is a comparative breeze!

现在，不要惊慌，如果您的网页播放器目前是10 MB。看来优化是艰巨的，但结果是一点点的努力通常是很容易的以这种方式结构化你的游戏照。想想每个个别场景上述步骤。如果你做的游戏中，你已经做了困难的部分。构建解决此加载的概念是一个比较轻微的一些场面！

If you open the console log (**Open Editor Log** button in the Console window ) after or during a build, you can see the size of each individual scene file. The console will show something like this:

如果你打开控制台日志（打开编辑器登录到控制台窗口按钮）后或在编译过程中，你可以看到每个=的场景文件的大小。控制台会显示这样的：

\*\*\*Player size statistics\*\*\*

Level 0 'Main Menu' uses 95.0 KB compressed.

Level 1 'Character Builder' uses 111.5 KB compressed.

Level 2 'Level 1' uses 592.0 KB compressed.

Level 3 'Level 2' uses 2.2 MB compressed.

Level 4 'Level 3' uses 2.3 MB compressed.

Total compressed size 5.3 MB. Total decompressed size 9.9 MB.

This game could use a little more optimization! For more information, we recommend you read the [reducing file size page](http://unity3d.com/support/documentation/Manual/Reducing%20File%20size.html).

此游戏可以用多一点的优化！欲了解更多信息，我们建议你阅读减少文件大小页面。

**The Most Important Steps最重要的步骤**

1. Load the menu first. Showing an animated logo is a great way to make time go by unnoticed, thus letting the download progress further. 第一加载菜单。显示的动画图案是一个伟大的方法使被忽视时间，从而让更多的下载进度。
2. Make the first level be short and not use a lot of assets. This way, the first level can be loaded quickly, and by keeping the player occupied playing it for a minute or two you can be sure that the download of all remaining assets can be completed in the background. Why not have a mini tutorial level where the user can learn the controls of the game? No reason for high-res textures here or loads of objects, or having all your enemies in the first level. Use the one with the lowest poly-count. And yes, this means you might have to design your game with the web player experience in mind.

使第一级是短的而不是使用大量的资产。这样，第一级可以加载迅速，并保持玩家播放了两分钟，你可以肯定，所有剩余资产可以在后台下载完成它。为什么没有一个小教程水平，用户可以了解游戏的控制？没有高清晰度的纹理在这里或物体负载，或有所有你在第一级的敌人的原因。使用最低的聚数之一。是的，这意味着你可能要设计时考虑到网站的玩家体验游戏。

1. There is no reason why all music must be available when the game starts. Externalize the music and load it via the [WWW](http://unity3d.com/support/documentation/ScriptReference/WWW.html) class. Unity compresses audio with the high quality codec, Ogg Vorbis. However even when compressed, audio takes up a lot of space, and if you have to fit things into 3 MB, if you have 5 minutes of music all the compression in the world won't save you. Sacrifices are needed. Load a very short track that you can loop until more music has been downloaded. Only load more music when the player is hooked on the first level.

没有任何理由将所有的音乐必须是可用的当游戏启动时。外部化的音乐装载通过WWW类。Unit使用高质量的编解码器，Ogg Vorbis来压缩音频。然而即使当压缩时，音频占用很大的空间，如果你要适应3 MB的事情，如果你有5分钟的音乐都在世界上的压缩不会拯救你。牺牲是必要的。装载很短的磁道，你可以循环，直到更多的音乐已被下载。当播放器在第一级挂钩时才装载更多的音乐。

1. Optimize your textures using their Import Settings. After you externalize music, textures easily take up 90% of the game. Typical texture sizes are too big for web deployment. In a small browser window, sometimes big textures don't even increase the visual fidelity at all. Make sure you use textures that are only as big as they must be (and be ready for more sacrifices here). Halving the texture resolution actually makes the texture size a quarter of what it was. And of course all textures should be DXT compressed.

优化你的纹理使用他们的导入设置。在具体化音乐之后，纹理很容易地利用了90％的游戏。典型的纹理尺寸过大对于web的部署。在一个小的浏览器窗口，大纹理有时甚至不增加视觉保真度的。请确保你使用大的纹理是他们必须的（并准备在这里更多的牺牲）。减半的纹理决议实际上使纹理尺寸的四分之一。当然，所有的纹理应DXT压缩。

1. Generally reduce the size of your web players. There is a manual page committed to the utilities Unity offers for optimizing file size [here](http://unity3d.com/support/documentation/Manual/Reducing%20File%20size.html). Although Unity uses cutting edge LZMA-based compression which usually compresses game data to anywhere from one half to a third of the uncompressed size, you'll need to try everything you can.

一般来说减少你的web播放器的大小。有一个手册页致力于公用工程提供unity优化文件大小在这里。虽然Unity采用先进的压缩算法的压缩，通常从一个地方压缩一半游戏数据给一个未压缩规模的三分之一，你需要尝试一切。

1. Try to avoid Resources.Load. While Resources.Load can be very handy, Unity will not be able to order your assets by when they are first used when you use Resources.Load, because any script could attempt to load the Resource. You can set which level will include all assets that can be loaded through Resources.Load in the **Edit->Project Settings->Player** using the **First Streamed Level With Resources** property. Obviously you want to move Resources.Load assets as late as possible into the game or not use the feature at all.

尽量避免Resources.Load。虽然Resources.Load可以非常方便的，Unity将无法整理第一次使用Resources.Load的资产，因为任何脚本可以尝试加载资源。你可以设置该级别将包括所有资产加载可以通过在e **Edit->Project Settings->Player里**使用的资源级别流媒体的第一属性。显然，你要移动Resources.Load资产尽可能晚进入游戏或根本不使用该功能。

**Publishing Streaming Web Players 发布流web播放器**

Streaming in Unity is level based, and there is an easy workflow to set this up. Internally, Unity does all the dirty work of tracking assets and organizing them in the compressed data files optimally, ordering it by the first scene that uses them. You simply have to ensure that the first levels in the Build Settings use as few assets as possible. This comes naturally for a "menu level", but for a good web experience you really need make sure that the first actual game levels the player is going to play are small too.

在unity里的流是以级别为基础的，有一个简单的工作流程处理此问题。在内部，Unity完成所有的资产跟踪和组织数据文件压缩优化他们最佳的工作，排序它通过第一场景使用他们。你只需确保在生成设置使用第一个级别尽可能少的资产。这种顺其自然的“一级菜单”，但有一个良好的WEB体验，你真的需要确保第一个实际的游戏等级播放器将播放小。

In order to use streaming in Unity, you select **Web Player Streamed** in the Build Settings. Then the content automatically starts as soon as all assets used by the first level are loaded. Try to keep the "menu level" to something like 50-100 KB. The stream continues to load as fast as it can, and meanwhile live decompresses. When you look at the Console during/after a build, you can see how large

为了使用流在Unity里，你选择Web播放器进行播放在生成设置里。然后内容将自动启动，尽快由第一级使用的所有资产被加载。尽量保持“一级菜单”50-100 KB喜欢的东西。流继续加载的速度，因为它可以同时现场解压。当你看到控制台在后台编译过程中，你可以看到有多大。

You can query the progress of the stream by level, and once a level is available it can be loaded. Use [GetStreamProgressForLevel](http://unity3d.com/support/documentation/ScriptReference/Application.GetStreamProgressForLevel.html) for displaying a progress bar and [CanStreamedLevelBeLoaded](http://unity3d.com/support/documentation/ScriptReference/Application.CanStreamedLevelBeLoaded.html) to check if all the data is available to load a specific level.

你可以通过级查询别流的处理，并且一旦一个级别是可以装载可用的。使用GetStreamProgressForLevel用于显示一个进度条和CanStreamedLevelBeLoaded检查是否所有数据都可以加载一个特定的级别。

This form of streaming is of course linear, which matches how games work in most cases. Sometimes that's not enough. Therefore Unity also provides you with API's to load a .unity3d file manually using the [WWW](http://unity3d.com/support/documentation/ScriptReference/WWW.html) class. Video and audio can be streamed as well, and can start playing almost immediately, without requiring the movie to be downloaded first. Finally Textures can easily be downloaded via the [WWW](http://unity3d.com/support/documentation/ScriptReference/WWW.html) class, as can any textual or binary data your game might depend on.

这种形式的流是当然的线性，以配合游戏如何在大多数情况下工作。有时那是不够的。因此，Unity还提供了API的加载。unity3d手动文件使用WWW类。视频和音频也可以被流化，几乎可以立即开始播放，而不需要电影，首先要下载电影。最后，纹理可以很容易地通过WWW类下载，这可以任何文本或二进制数据取决于你的游戏。

[**Unity Manual**](http://unity3d.com/support/documentation/Manual/index.html) > [**Advanced**](http://unity3d.com/support/documentation/Manual/Advanced.html) > [**Web Player Deployment**](http://unity3d.com/support/documentation/Manual/Web%20Player%20Deployment.html)

Unity手册->高级->web播放器部署

**Web Player Deployment WEB播放器部署**

When building a **Web Player**, Unity automatically generates an HTML file next to the player data file. It contains the default HTML code to load the web player data file using both the Netscape plugin and ActiveX Control.

当创建一个WEB播放器时，Unity自动产生一个HTML文件紧接着来到播放器数据文件。它包括HTM默认的代码去装载WEB播放器数据文件使用Netscape插件和ActiveX控制。

It is possible to further tweak and customize the generated HTML file to make it fit better with the containing site's design, to add more HTML content, etc. The following pages discuss the related subjects in depth:

它有可能进一步调整和定制生成的HTML文件，使之更好地适应包含站点的设计，增加更多的HTML内容，等等。以下页面深入讨论有关议题：

* [HTML code to load Unity content](http://unity3d.com/support/documentation/Manual/HTML%20code%20to%20load%20Unity%20Web%20Player%20content.html) HTML代码装入Unity内容
* [Customizing the Unity Web Player loading screen](http://unity3d.com/support/documentation/Manual/Customizing%20the%20Unity%20Web%20Player%20loading%20screen.html) 自定义Unity WEB播放器装入屏幕
* [Customizing the Unity Web Player's Behavior](http://unity3d.com/support/documentation/Manual/WebPlayerBehaviorTags.html) 自定义Unity WEB播放器行为
* [Unity Web Player and browser communication](http://unity3d.com/support/documentation/Manual/Unity%20Web%20Player%20and%20browser%20communication.html) Unity WEB播放器和浏览通信
* [Detecting the Unity Web Player using browser scripting](http://unity3d.com/support/documentation/Manual/Detecting%20the%20Unity%20Web%20Player%20using%20browser%20scripting.html) 使用浏览脚本检测Unity WEB播放器
* [Publishing active content](http://unity3d.com/support/documentation/Manual/Publishing%20active%20content.html) 发布活动内容

[**Unity Manual**](http://unity3d.com/support/documentation/Manual/index.html) > [**Advanced**](http://unity3d.com/support/documentation/Manual/Advanced.html) > [**Web Player Deployment**](http://unity3d.com/support/documentation/Manual/Web%20Player%20Deployment.html) > [**HTML code to load Unity content**](http://unity3d.com/support/documentation/Manual/HTML%20code%20to%20load%20Unity%20Web%20Player%20content.html)

Unity手册->高级->web播放器部署->HTML代码装入Unity内容

**HTML code to load Unity content** HTML代码装入Unity内容

**Web Player** content is loaded in the browser using the Unity Web Player's plugin or ActiveX control. The HTML code required to load and display web player content is different for Microsoft's Internet Explorer compared to all other browsers. This is because Internet Explorer uses ActiveX Controls and therefore an **object** tag must be used whereas all other supported browsers use plugins and therefore an **embed** tag must be used instead.

Web播放内容被加载在浏览器中使用Unity的Web Player的插件或ActiveX控件。所需的HTML代码加载和显示Web播放器的内容是不同的对于微软IE浏览器与所有其他浏览器比较而言。这是因为Internet Explorer使用ActiveX控件和标签的对象，必须使用而所有其他支持的浏览器支持的插件并且因此内嵌标记必须替代使用。

The HTML file generated by Unity when building a web player contains all the commonly required functionality - it does Unity plugin detection, enables easy web player installation, and supports all browsers that can run Unity Web Player. In most cases you don't have to modify the HTML file at all. This document describes the inner workings of HTML code required for Unity content.

该HTML文件通过Unity生成当创建一个web播放器包含了所有的公共需要的功能-它做Unity的插件检测，启用简单的web播放器安装，并支持所有的浏览器，可以运行Unity的Web播放器。在大多数情况下，你不必修改HTML文件的。本文档描述的HTML代码为Unity的内容进行内部工作。

The minimal HTML required to load Unity Web Player content in all supported browsers is:

所需的最小的HTML加载在所有支持Unity的Web浏览器播放的内容是：

<object id="UnityObject" classid="clsid:444785F1-DE89-4295-863A-D46C3A781394"

width="600" height="450"

codebase="http://webplayer.unity3d.com/download\_webplayer/UnityWebPlayer.cab#version=2,0,0,0">

<param name="src" value="MyDataFile.unity3d" />

<embed id="UnityEmbed" src="MyDataFile.unity3d" width="600" height="450"

type="application/vnd.unity" pluginspage="http://www.unity3d.com/unity-web-player-2.x" />

</object>

In the above example the **object** and **embed** tags are used in such a way that Internet Explorer uses the object tag and ignores the embed tag, while all other browsers use the embed tag and ignore the object tag.

在上面的例子里的object和embed标签中使用这样的方式，Internet Explorer使用的object标记，而忽略embed标签的，而所有其他浏览器使用的标签，而忽略嵌入object标签。

In both the object and embed tags there are various parameters that control how the web player content is displayed:

在这两个object和embed标签里有各种参数控制web播放器内容如何显示：

* **src**: Defines the path to the web player data file to be displayed.
* **src**：定义路径web播放器数据文件被显示。
* **height**, **width**: Defines the display size for the web player content. The values provided can be pixel values as expressed in the example above (600 pixels wide and 450 pixels tall) or as percentage values (width="100%" for example).
* 高度，宽度：定义的WEB播放器内容显示的大小。所提供的值可在上面的例子表示像素值（600像素宽，450像素高）或百分比值（宽度=“100％”的例子）。
* **id**: Can be used by external browser scripts in order to communicate with the web player content. The default id values included when publishing web player content are **UnityObject** for the object tag and **UnityEmbed** for the embed tag, but those can be changed to different values as necessary.
* id：可使用外部浏览器的脚本以便沟通web播放器的内容。默认ID值包括发布web播放器的内容是UnityObject为object的标签和UnityEmbed为embed标签，但这些是可以改变的，以在必要时不同的价值需要。

The other parameters found in the object and embed tags above are required and should not be modified. The **codebase** and **pluginspage** parameters point to the location from which the Unity Web Player can be installed. The **classid** parameter is a unique identifier for the Unity Web Player ActiveX Control and the **type** parameter is the content mime-type for the Unity Web Player plugin.

在上述的object和embed标签里发现其它参数是必需的而且不应该被修改。基本代码插件页的参数指出在开始Unity的Web播放器可以安装的位置。参数的ClassID是一个Unity的Web Player ActiveX控件和类型参数的唯一标识符是内容的MIME的Unity WEB Player插件类型。

[**Unity Manual**](http://unity3d.com/support/documentation/Manual/index.html)> [**Advanced**](http://unity3d.com/support/documentation/Manual/Advanced.html)> [**Web Player Deployment**](http://unity3d.com/support/documentation/Manual/Web%20Player%20Deployment.html) > [**Customizing the Unity Web Player loading screen**](http://unity3d.com/support/documentation/Manual/Customizing%20the%20Unity%20Web%20Player%20loading%20screen.html)

Unity手册->高级->web播放器部署->自定义Unity WEB播放器装入屏幕

**Customizing the Unity Web Player loading screen**

自定义Unity WEB播放器装入屏幕

By default the Unity **Web Player** displays a small Unity logo and a progress bar while loading web player content. It is possible to customize the appearance of that loading screen, including both the logo and progress bar display.

默认情况下，Unity Web播放器显示一个小Unity标识和进度条，而载入网页播放器的内容。它可以自定义的装载屏幕的外观，包括徽标和进度条显示。

Please note that modifying the loader images is only possible with **Unity Pro**.

请注意，修改加载图像，仅Unity专业版具有。

There are six optional **object** and **embed** tag parameters that can be used to customize the appearance of the Unity Web Player loading screen. Those optional tag parameters are:

有6个可选的object和embed标签，可用于自定义的Unity Web播放器加载屏幕外观参数。这些可选标记的参数是：

* **backgroundcolor**: The background color of the web player content display region during loading, the default is white.
* 背景颜色：在加载的网页内容显示区域播放背景颜色，默认为白色。
* **bordercolor**: The color of the one pixel border drawn around the web player content display region during loading, the default is white.
* 边框颜色：一个像素的边框颜色绘制围绕Web播放器内容显示区域在装载过程中，默认为白色。
* **textcolor**: The color of error message text (when data file fails to load for example). The default is black or white, depending on the background color.
* 文字颜色：该错误消息的文本颜色（当数据文件加载失败的例子）。默认的是黑色或白色，依据背景颜色。
* **logoimage**: The path to a custom logo image, the logo image is drawn centered within the web player content display region during loading.
* 徽标图像：自定义徽标图像路径，徽标图像绘制在web播放器的内容显示在装载地区为中心。
* **progressbarimage**: The path to a custom image used as the progress bar during loading. The progress bar image's width is clipped based on the amount of file loading completed, therefore it starts with a zero pixel width and animates to its original width when the loading is complete. The progress bar is drawn beneath the logo image.
* 装载进度条：在自定义的图像在装载进度条使用的路径。进度条图片的宽度被剪辑基于文件的装载量完成，因此，它开始以零像素的宽度和动画去它自己的宽度当加载完成时。进度条被绘制下方标志图片。
* **progressframeimage**: The path to a custom image used to frame the progress bar during loading.
* 进度帧图像：在自定义图像路径用于装载过程中的进度条。

All color values provided must be 6-digit hexidecimal colors, (eg. FFFFFF, 020F16, etc.). The image paths provided can be either relative or absolute links and all image files must be RGB (without transparency) or RGBA (with transparency) 8-bit/channel PNG files. Finally, the **progressframeimage** and the **progressbarimage** should be the same height.

所有颜色值提供必须是6位十六进制颜色，（例如：000000，020F16等）。所提供的图像路径可以是相对或绝对的连接，所有的图像文件必须是RGB（不透明度）或RGBA（透明）8-位/通道 PNG文件。最后，进度帧图像和进度条图像应该是相同的高度。

Here is an example set of object and embed tags that customize the appearance of the Unity Web Player loading screen. The background color is set to light gray (**A0A0A0**), border color to black (**000000**), text color to white (**FFFFFF**) and loader images to **MyLogo.png**, **MyProgressBar.png** and **MyProgressFrame.png**.

这里是一个例子设置object和embed标签自定义Unity Web播放器加载屏幕的外观。背景色设置为浅灰色（A0A0A0），边框颜色为黑色（000000），文本颜色为白色（000000）和装载图像为MyLogo.png，MyProgressBar.png和MyProgressFrame.png。

<object id="UnityObject" classid="clsid:444785F1-DE89-4295-863A-D46C3A781394"

width="600" height="450"

codebase="http://webplayer.unity3d.com/download\_webplayer/UnityWebPlayer.cab#version=2,0,0,0">

<param name="src" value="MyDataFile.unity3d" />

<param name="backgroundcolor" value="A0A0A0" />

<param name="bordercolor" value="000000" />

<param name="textcolor" value="FFFFFF" />

<param name="logoimage" value="MyLogo.png" />

<param name="progressbarimage" value="MyProgressBar.png" />

<param name="progressframeimage" value="MyProgressFrame.png" />

<embed id="UnityEmbed" src="MyDataFile.unity3d" width="600" height="450"

type="application/vnd.unity" pluginspage="http://www.unity3d.com/unity-web-player-2.x"

backgroundcolor="A0A0A0" bordercolor="000000" textcolor="FFFFFF" logoimage="MyLogo.png"

progressbarimage="MyProgressBar.png" progressframeimage="MyProgressFrame.png" />

</object>

Notice that in the above example, the optional tag parameters are included in both the **object** and **embed** tag blocks as name/value pairs. It is important to include all tag parameters in both locations in order to ensure that all users see the same loading screen regardless of the browser they are using.

请注意，在上面的例子中，可选的标签参数均包含object和embed标签块作为对应的名称/值。重要的是包括在这两个地点的所有标签的参数，以确保所有用户看到相同的加载屏幕无论他们使用的浏览器。

**Note**: developers that modify the default HTML file created by Unity must be sure to make edits in a few locations due to how that HTML file is structured. The first section to modify is the JavaScript block that detects the Unity Web Player, and if found writes out the **object** and **embed** tags. The second section to modify is the **noscript** block at the end in case the page is viewed by a user with JavaScript disabled. Editing both sections, and both the **object** and **embed** tags in each section, will ensure that all users, in all browsers will have the same experience.

注：开发人员修改默认的HTML文件的通过Unity创造的必须确保作出应有怎样的HTML文件的结构是少数部分的修改。第一部分修改是JavaScript块检测的Unity Web播放器，如果发现写出的object和embed标签。第二部分修改，是在底部的NoScript的情况下，页面是由用户查看一个禁用JavaScript。编辑两部分，两者的object和embed标签在每一节里，将确保所有用户，在所有浏览器都会有相同的体验。

[**Unity Manual**](http://unity3d.com/support/documentation/Manual/index.html) > [**Advanced**](http://unity3d.com/support/documentation/Manual/Advanced.html) > [**Web Player Deployment**](http://unity3d.com/support/documentation/Manual/Web%20Player%20Deployment.html) > [**Customizing the Unity Web Player's Behavior**](http://unity3d.com/support/documentation/Manual/WebPlayerBehaviorTags.html)

Unity手册->高级->web播放器部署->自定义Unity WEB播放器的行为

**Customizing the Unity Web Player's Behavior**自定义Unity WEB播放器的行为

The Unity **Web Player** allows developers to use a few optional **object** and **embed** tag parameters to easily control its behavior in a few ways:

在Unity 的Web Player允许开发人员使用一些可选的object和embed标签的参数去易于控制其在数个方面的行为：

* **disableContextMenu**: This parameter controls whether or not the Unity **Web Player** displays a context menu when the user right- or control-clicks on the content. Setting it to true prevents the context menu from appearing and allows content to utilize right-mouse behavior, the default is false.
* 禁用上下文菜单：此参数控制是否Unity Web播放器显示上下文菜单当用户权限或控制的内容点击时。将它设置为true，防止出现上下文菜单，允许内容利用右鼠标行为，默认为false。
* **disableExternalCall**: This parameter controls whether or not the Unity **Web Player** allows content to communicate with browser-based JavaScript. Setting it to true prevents browser communication and so content cannot call or execute JavaScript in the browser, the default is false.
* 禁用外部调用：此参数控制是否Unity Web播放器允许内容沟通基于浏览器的JavaScript。设置为true防止浏览器的通信而且内容不能调用，或在浏览器中执行JavaScript的，默认为false。
* **disableFullscreen**: This parameter controls whether or not the Unity **Web Player** allows content to be viewed in fullscreen mode. Setting it to true prevents fullscreen viewing and removes the "Go Fullscreen" entry from the context menu, the default is false.
* 禁用全屏：此参数控制是否Unity Web播放器允许内容将在全屏模式浏览。设置为true防止全屏查看和删除“到全屏”从上下文菜单项，默认为false。

Here is an example set of **object** and **embed** tags that limit the behavior of the Unity **Web Player**, the context menu will not be shown and fullscreen viewing is not allowed:

这里是object和embed标签设置，限制了Unity web 播放器的行为，在上下文菜单将不会被显示和全屏观看是不允许的：

<object id="UnityObject" classid="clsid:444785F1-DE89-4295-863A-D46C3A781394"

width="600" height="450"

codebase="http://webplayer.unity3d.com/download\_webplayer/UnityWebPlayer.cab#version=2,0,0,0">

<param name="src" value="MyDataFile.unity3d" />

<param name="disableContextMenu" value="true" />

<embed id="UnityEmbed" src="MyDataFile.unity3d" width="600" height="450"

type="application/vnd.unity" pluginspage="http://www.unity3d.com/unity-web-player-2.x"

disableContextMenu="true" />

</object>

In the above example you'll notice that neither **disableExternalCall** nor **disableFullscreen** are specified, therefore their default values are used. You'll also notice that **disableContextMenu** is specified and included in both the **object** and **embed** tag blocks. It is important to include all tag parameters in both locations in order to ensure that all users are offered the same functionality regardless of the browser they are using.

在上面的例子中，你会发现，既不是disableExternalCall也不是disableFullscreen指定，因此，他们将使用默认值。你还会发现disableContextMenu被指定以及包含在object和embed标签块中。重要的是包括在这两个标签的参数，以确保所有用户提供相同的功能而不管他们使用的浏览器。

**Note**: developers that modify the default HTML file created by Unity must be sure to make edits in a few locations due to how that HTML file is structured. The first section to modify is the JavaScript block that detects the Unity Web Player, and if found writes out the **object** and **embed** tags. The second section to modify is the **noscript** block at the end in case the page is viewed by a user with JavaScript disabled. Editing both sections, and both the **object** and **embed** tags in each section, will ensure that all users, in all browsers will have the same experience.

注：开发人员修改默认的HTML文件的通过Unity创造的必须确保作出应有怎样的HTML文件的结构是少数部分的修改。第一部分修改是JavaScript块检测的Unity Web播放器，如果发现写出的object和embed标签。第二部分修改，是在底部的NoScript的情况下，页面是由用户查看一个禁用JavaScript。编辑两部分，两者的object和embed标签在每一节里，将确保所有用户，在所有浏览器都会有相同的体验。

[**Unity Manual**](http://unity3d.com/support/documentation/Manual/index.html) > [**Advanced**](http://unity3d.com/support/documentation/Manual/Advanced.html) > [**Web Player Deployment**](http://unity3d.com/support/documentation/Manual/Web%20Player%20Deployment.html) > [**Unity Web Player and browser communication**](http://unity3d.com/support/documentation/Manual/Unity%20Web%20Player%20and%20browser%20communication.html)

Unity手册->高级->web播放器部署-> Unity WEB播放器和浏览器通信

**Unity Web Player and browser communication** Unity WEB播放器和浏览器通信

The HTML page that contains **Unity Web Player** content can communicate with that content and vice versa. Basically there are two communication directions:

HTML页面，其中包含Unity Web播放的内容可以通信的内容，反之亦然。基本上有两种通信方向：

* The web page calls functions inside the Unity web player content.
* 该网页内调用Unity web播放器内容内部的功能。
* The Unity web player content calls functions in the web page.
* Unity web播放器的内容调用功能在在web页中。

Each of these communication directions is described in more detail below.

这些通信方向的每一个详细描述如下。

**Calling Unity web player content functions from the web page**

**从web页面调用Unity web播放器内容**

The Unity Web Player plugin and ActiveX Controls both have a function, **SendMessage()**, that can be called from a web page in order to call functions within Unity web player content. This function is very similar to the [GameObject.SendMessage](http://unity3d.com/support/documentation/ScriptReference/GameObject.SendMessage.html) function in the Unity scripting API. When called from a web page you pass an object name, a function name and a single argument, and **SendMessage()** will call the given function in the given game object.

在Unity 的Web Player插件和ActiveX控件都有一个函数，SendMessage()，可以从网页上一个web页面被调用，为了调用Unity web播放器内容内部功能。这个功能是非常类似于GameObject.SendMessage函数在Unity 脚本API里。当从所谓的网页上你传递一个对象的名称，一个函数名和一个简单参数，和SendMessage()将调用给定的函数在在给定对象的游戏里。

In order to call the Unity Web Player's **SendMessage()** function you must first get a reference to the Unity web player content object being displayed. You can use JavaScript's **document** object and its **getElementById()** function to obtain a reference to the content. Here is an example JavaScript function that would execute the **SendMessage()** function on the Unity web player content with an object/embed tag id value of **UnityContent**; in turn **SendMessage()** will then call the function **MyFunction()** on the game object named *MyObject*, passing a piece of string data as an argument:

为了调用Unity Web播放器的SendMessage()函数必须先得到网站的Unity 播放器内容对象的一个引用被显示出来。你可以使用JavaScript的文档对象和getElementById()函数来获取对内容的引用。下面是一个JavaScript示例函数将利用object/ embed标签的UnityContent id值在Unity web播放器内容上执行SendMessage（）函数，然后反过来SendMessage（）将会调用函数调用MyFunction（）在游戏对象名称上的MyObject来传递的一个字符串数据作为参数：

<script type="text/javascript" language="javascript">

<!--

function SaySomethingToUnity()

{

document.getElementById("UnityContent").SendMessage("MyObject", "MyFunction", "Hello from a web page!");

}

-->

</script>

Inside of the Unity web player content you need to have a script attached to the **GameObject** named **MyObject**, and that script needs to implement a function named **MyFunction**:

在Unity 网络播放器内容的内部里你需要一个脚本附加到名为MyObject的GameObject上，该脚本需要实现名为myFunction函数：

function MyFunction(param : String)

{

Debug.Log(param);

}

A single string, integer or float argument must be passed when using **SendMessage()**, the parameter is required on the calling side. If you don't need it then just pass a zero or other default value and ignore it on the Unity side. Additionally, the game object specified by the name can be given in the form of a path name. For example, **/MyObject/SomeChild** where **SomeChild** must be a child of **MyObject** and **MyObject** must be at the root level due to the '/' in front of its name.

一个单一的字符串，整数或浮点数必须通过使用SendMessage()传递，参数是需要在非正式的调用。如果你不需要它然后只通过一个零或其他默认值并忽略它在Unity方面。此外，游戏对象指定通过名称可以得到在一个路径名。例如，/ MyObject/ SomeChild在那儿SomeChild必须是MyObject的子和MyObject必须在根级别由于'/在其名称的前面。

The default html file generated when you publish web player content includes both an object and embed tag in order to have the content load properly in all browsers. In order to allow browser-based JavaScript to distinguish between the two tag elements they each use a unique id value, **UnityObject** for the object tag and **UnityEmbed** for the embed tag. Because of this, the default html file also includes a JavaScript function, **GetUnity()**, that performs some simple browser detection and returns a reference to the tag element in use. Here is an example using that function:

默认的HTML文件生成当你发布web播放器的内容时包括了一个object和embed标签，以便在所有浏览器中的内容正确加载。为了使基于浏览器的JavaScript来区分这两种标记的元素，他们每次使用一个唯一的ID值，UnityObject为object标签和UnityEmbed为embed标签。正因为如此，默认的HTML文件还包括一个JavaScript函数，GetUnity()，即执行一些简单的浏览器检测并返回一个使用的参考标记元素。下面是一个示例使用该功能：

<script type="text/javascript" language="javascript">

<!--

function SaySomethingToUnity()

{

GetUnity().SendMessage("MyObject", "MyFunction", "Hello from a web page!");

}

-->

</script>

**Calling web page functions from Unity web player content**

从Unity播放器内容调用web页函数

In order to call a web page function from within your Unity web player content you must use the [**Application.ExternalCall()**](http://unity3d.com/support/documentation/ScriptReference/Application.ExternalCall.html) function. Using that function you can call any JavaScript function defined in the web page, passing any number of parameters to it. Here is an example Unity script that uses the **Application.ExternalCall()** function to call a function named **SayHello()** found within the web page, passing a piece of string data as an argument:

为了从你内部Unity web播放器内容调用一个WEB页函数，你必须使用Application.ExternalCall（）函数。使用该功能，你可以调用任何JavaScript函数中定义的网页，传递任意数量的参数给它。这里有一个例子Unity 脚本使用Application.ExternalCall（）函数来调用一个函数名为SayHello（）发现的网页，传递一个字符串数据作为参数：

Application.ExternalCall( "SayHello", "The game says hello!" );

The web page would need to define the **SayHello()** function, for example:

该网页将需要定义sayHello()函数，例如：

<script type="text/javascript" language="javascript">

<!--

function SayHello( arg )

{

// show the message

alert( arg );

}

-->

</script>

**Executing arbitrary browser code from Unity web player content**

从Unity web 播放器内存执行任意浏览器的代码。

You don't even have to define functions in the embedding web page, instead you can use the [**Application.ExternalEval()**](http://unity3d.com/support/documentation/ScriptReference/Application.ExternalEval.html) function to execute arbitrary browser code from the web player content.

你甚至不必在嵌入网页定义功能，替代的是你可以使用Application.ExternalEval（）函数来执行任意浏览器的代码从网页播放器的内容。

The following example checks that the page embedding the web player content is fetched from a certain host (unity3d.com), if that's not the case then it will redirect to another URL. This technique can be used to prevent deep linking to your web player content:

下面的例子检查该网页嵌入web播放器的内容是从某主机（unity3d.com），如果不是这样，那么它将被重定向到另一个URL。这种技术可以用来防止深层链接到你的web播放器的内容：

Application.ExternalEval(

"if(document.location.host != 'unity3d.com') { document.location='http://unity3d.com'; }"

);

[**Unity Manual**](http://unity3d.com/support/documentation/Manual/index.html)>[**Advanced**](http://unity3d.com/support/documentation/Manual/Advanced.html)>[**Web Player Deployment**](http://unity3d.com/support/documentation/Manual/Web%20Player%20Deployment.html)>[**Detecting the Unity Web Player using browser scripting**](http://unity3d.com/support/documentation/Manual/Detecting%20the%20Unity%20Web%20Player%20using%20browser%20scripting.html)

Unity手册->高级->web播放器部署->使用浏览脚本检测Unity WEB播放器

**Detecting the Unity Web Player using browser scripting**

**使用浏览脚本检测Unity WEB播放器**

Any time you build and post **Web Player** content you should take steps to ensure the end-user has the Unity Web Player before attempting to display the content. While the default HTML file generated when you publish web player content contains a **noembed** tag for the cases where the end-user doesn't have the player, that tag isn't used or displayed consistently across all browsers and fails to offer a consistent and predictable user experience. Therefore instead of relying on the browser's handling of the noembed tag you can use browser script to determine whether the end-user has the Unity Web Player installed and respond appropriately.

任何时候你建立和发送web播放器内容，你应该采取步骤，确保最终用户有Unity Web播放器在尝试显示内容之前。虽然默认的HTML文件生成当你发布web播放器内容时包含一个noembed标签的情况下最终用户那里没有播放器，该标记是不被使用或通过所有浏览器一贯的显示和不能提供一个一致的可预见的用户体验。因此，替代依赖浏览器的noembed的标签处理，你可以使用浏览器的脚本来确定是否最终用户拥有Unity Web播放器安装，并采取适当的响应。

Detecting the Unity Web Player in all browsers can be done using a combination of JavaScript and VBScript browser code. The process of detecting the web player involves the following steps:

在所有浏览器里检测Unity Web播放器可以做到使用一个JavaScript和VBScript浏览器代码组合。在检测web播放器包括以下步骤：

* Browser Detection: is the page being viewed on Windows using Microsoft Internet Explorer or some other browser?
* 浏览器检测：是正在浏览的网页上使用Microsoft Internet Explorer或其他浏览器窗口？
* ActiveX Control Detection: If the page is being viewed on Windows in Internet Explorer, is the web player's ActiveX Control installed?
* ActiveX控件检测：如果web 被浏览在Windows Internet Explorer中，它的网页播放器的ActiveX控件是是否安装？
* Plugin Detection: if the page is being viewed in any other browser, is the **mime type** understood and the plugin installed?
* 插件检测：如果网页被任何其他浏览器中查看，是mime类型理解和插件安装？

Here is an example JavaScript and VBScript function that performs the above steps in order to detect the Unity Web Player:

这里有一个例子JavaScript和VBScript函数执行上述步骤，以检测Unity Web播放器：

<script language='VBScript'>

function detectUnityWebPlayerActiveX

on error resume next

dim tControl, res, ua, re, matches, major

res = 0

set tControl = CreateObject("UnityWebPlayer.UnityWebPlayer.1")

if IsObject(tControl) then

if tControl.GetPluginVersion() = "2.5.0f5" then

' 2.5.0f5 on Vista and later has an auto-update issue

' on Internet Explorer. Detect Vista (6.0 or later)

' and in that case treat it as not installed

ua = Navigator.UserAgent

set re = new RegExp

re.Pattern = "Windows NT (\d+)\."

set matches = re.Execute(ua)

if matches.Count = 1 then

major = CInt(matches(0).SubMatches(0))

if major < 6 then

res = 1

end if

end if

else

res = 1

end if

end if

detectUnityWebPlayerActiveX = res

end function

</script>

<script language="javascript1.1" type="text/javascript">

function detectUnityWebPlayer () {

var tInstalled = false;

if (navigator.appVersion.indexOf("MSIE") != -1 &&

navigator.appVersion.toLowerCase().indexOf("win") != -1)

{

tInstalled = detectUnityWebPlayerActiveX();

}

else if (navigator.mimeTypes && navigator.mimeTypes["application/vnd.unity"])

{

if (navigator.mimeTypes["application/vnd.unity"].enabledPlugin &&

navigator.plugins && navigator.plugins["Unity Player"])

{

tInstalled = true;

}

}

return tInstalled;

}

</script>

When the function above is called, it checks for the **Unity Web Player** and returns a boolean value as a result. A return value of true indicates that the Unity Web Player is installed whereas a return value of false indicates that it is not. The HTML file generated by Unity when building web player contains a very similar function.

当上面的函数被调用，它检查Unity Web播放器并返回一个布尔值结果。返回的值为真表明Unity web播放器是安装的，而返回一个假值表明没有安装。当编译web播放器包含一个非常类似的功能时，通过Unity生成HTML文件。

The detection is performed separately for Internet Explorer and all the other browsers:

该检测是为Internet Explorer和所有其他的浏览器分别执行的：

* For IE, a VBScript snippet is used. VBScript is used instead of JavaScript because after detection it releases the plugin resource immediately (whereas in JavaScript it would only happen the next time garbage collection is performed). The script tries to create an ActiveX object named "UnityWebPlayer.UnityWebPlayer.1" which is the name of Unity Web Player ActiveX control.
* 对于IE，一个VBScript代码段使用。 VBScript是用来代替JavaScript，因为在检查它之后立即释放插件资源（而在JavaScript中仅发生下一次垃圾收集完成时）。该脚本尝试创建一个ActiveX对象名为“UnityWebPlayer.UnityWebPlayer.1”，这是Unity 的Web Player ActiveX控件的名称。
* For other browsers it is checked whether a MIME type "application/vnd.unity" is understood and plugin named "Unity Player" is installed and enabled.
* 对于其它浏览器，这是检查是否一个MIME类型“application/ vnd.unity”是理解以及插件名为“Unity player”是安装并启用。

Here is an example of using the function within a HTML page to detect the Unity Web Player and then respond appropriately:

下面是一个使用函数在一个HTML页内部的例子来检测Unity Web播放器以及然后作出适当的反应：

<script type="text/javascript" language="javascript">

<!--

-- check for the Unity Web Player

var tIsInstalled = detectUnityWebPlayer();

if (tIsInstalled)

{

// write the content object and embed tags

document.write("<object classid='clsid:444785F1-DE89-4295-863A-D46C3A781394' \");

document.write(" codebase='http://webplayer.unity3d.com/download\_webplayer/UnityWebPlayer.cab#version=2,0,0,0' \n");

document.write(" id='UnityObject' width='600' height='450' > \n");

document.write(" <param name='src' value='MyDataFile.unity3d' /> \n");

document.write(" <embed type='application/vnd.unity' pluginspage='http://www.unity3d.com/unity-web-player-2.x' \n");

document.write(" id='UnityEmbed' width='600' height='450' src='MyDataFile.unity3d' \n");

document.write(" /> \n");

document.write("</object>");

}

else

{

// write out a simple message prompting the user to install the Unity Web Player

document.write("<div align='center'> \n");

document.write(" This content requires the Unity Web Player,");

document.write(" please use the link below to install the player today:<br /><br />\n");

document.write(" <a href='http://www.unity3d.com/unity-web-player-2.x'> \n");

document.write(" Install the Unity Web Player \n");

document.write(" </a> \n");

document.write("</div> \n");

}

-->

</script>

As you can see the function is used to detect the Unity Web Player, then if the web player is found the required **object** and **embed** tags are written into the page. Note that if the web player is not found then a message is displayed prompting the user to install the Unity Web Player.

正如你所看到的函数是用来检测Unity Web播放器，那么如果web播放器找到所需要的object和embed标签被写入到页面中。请注意，如果web播放器没有找到，则显示一条消息，提示用户安装Unity Web播放器。

[**Unity Manual**](http://unity3d.com/support/documentation/Manual/index.html) > [**Advanced**](http://unity3d.com/support/documentation/Manual/Advanced.html) > [**Web Player Deployment**](http://unity3d.com/support/documentation/Manual/Web%20Player%20Deployment.html) > [**Publishing active content**](http://unity3d.com/support/documentation/Manual/Publishing%20active%20content.html)

Unity手册->高级->web播放器部署->发布活动内容

# Publishing active content 发布活动内容

When publishing active content, like Unity Web Player content, developers must consider how that content will be handled in various browsers. Specifically, developers need to consider how that content will be handled by Microsoft's Internet Explorer browser as it offers slightly different behavior than all other supported browsers today. Depending on how your content is embedded within your web page it may require the end-user to activate the content before it will be sensitive to mouse and keyboard input. In order to activate the content the end-user will have to either click once on the content itself, or click OK in a dialog box to load the ActiveX Control. While this doesn't prevent web player content from being used in Internet Explorer, it does offer a potentially confusing and less than optimal end-user experience. Fortunately, Microsoft has provided page design solutions so that developers can create or update their web pages in such a way that end-users do not have to click to activate web player content. Below you will find details on how to implement one of those solutions.

当发布活动内容时，像Unity的Web播放器内容，开发人员必须考虑如何将这些内容在不同的浏览器处理。具体来说，开发人员需要考虑如何将这些内容通过微软的IE浏览器处理的，因为今天它提供了稍微不同的比其他所有支持的浏览器行为。根据你的内容是如何在你的网页嵌入它可能需要最终用户激活的内容，然后将敏感的鼠标和键盘输入。为了激活内容，最终用户将不得不或者单击一次在对话框中的内容本身，或单击确定以加载ActiveX控件。虽然这并不妨碍在Internet Explorer中使用网络播放的内容，它提供了一个潜在的混乱，不是最佳的最终用户体验。幸运的是，微软提供了网页设计解决方案，使开发人员可以创建或更新这些企业的网页，最终用户不需要点击激活web播放器的内容。下面你会发现就如何落实这些解决方案之一的细节。

For more information on activating ActiveX Controls in Internet Explorer please read [Activating ActiveX Controls](http://msdn.microsoft.com/library/default.asp?url=/workshop/author/dhtml/overview/activating_activex.asp).

欲了解更多有关激活的Internet Explorer ActiveX控件信息，请阅读激活ActiveX控件。

In order to prevent your end-user from having to activate your Unity Web Player content prior to using it you must use an external JavaScript file to dynamically write the required object tags into your web page at run-time. In order to keep things simple, the solution outlined here will use an external JavaScript file to write both the object and embed tags so the page loads and plays the same in all browsers. In the simplest case you would create an external JavaScript file with a function that when called, writes out the required object and embed tags using document.write() calls.

为了防止你的最终用户不必激活Unity的Web播放器内容使之前用它，你必须使用一个外部JavaScript文件去动态写入所需要的object标签到你的网页里在运行时。为了简单起见，这里列出的解决方案将使用一个外部JavaScript文件写入object及embed标签，以便在页面加载和播放相同在所有的浏览器里。在最简单的情况下，你将创建一个具有函数调用的外部JavaScript文件，编写出需要object和embed标签使用document.write（）调用。

Here is an example JavaScript function that uses document.write() calls to dynamically create the necessary object and embed tags:

下面是一个JavaScript示例函数，使用document.write（）调用动态创建必要的object和embed标签：

<script type="text/javascript" language="javascript">

function writeUnityTags (aSrc, aWidth, aHeight) {

// write the content object and embed tags

document.write("<object ");

document.write(" classid='clsid:444785F1-DE89-4295-863A-D46C3A781394' \n");

document.write(" codebase='http://webplayer.unity3d.com/download\_webplayer/UnityWebPlayer.cab#version=2,0,0,0' \n");

document.write(" id='UnityObject' width='" + aWidth + "' height='" + aHeight + "' > \n");

document.write(" <param name='src' value='" + aSrc + "' /> \n");

document.write(" <embed type='application/vnd.unity' pluginspage='http://www.unity3d.com/unity-web-player-2.x' \n");

document.write(" id='UnityEmbed' width='" + aWidth + "' height='" + aHeight + "' src='" + aSrc + "' \n");

document.write(" /> \n");

document.write("</object\>");

}

</script>

Notice that when the function is called you must pass three arguments, those arguments being the path to the web player data file as well as the width and height of the content to be displayed. Using those arguments the function then writes out the object and embed tags directly into the web page that called the function.

注意，当该函数被调用，您必须传递3个参数，这些参数作为web播放器的数据文件的路径，以及宽度和高度的内容显示。使用这些参数的函数，然后写出到网页调用该函数的object和直接embed标签。

Here is an example of using the writeUnityTags() JavaScript function in a web page, where that function is defined in an external JavaScript file named unityweb.js:

下面是一个使用在一个网页里使用writeUnityTags（）JavaScript函数的例子，那个函数被定义子一个外部JavaScript文件名为unityweb.js：

<script type="text/javascript" language="javascript" src="unityweb.js"></script>

<script type="text/javascript" language="javascript">

// call the writeUnityTags function

writeUnityTags("MyFile.unity3d", 640, 480);

</script>

Using an external JavaScript file with a simple function whose sole purpose is to write out the object and embed tags is sufficient to avoid requiring your end-user to activate the content prior to using and interacting with it. Additionally the above solution functions properly in all browsers and therefore implementing it ensures that the end-user experience is consistent regardless of their browser and platform combination.

使用一个简单的函数，其唯一目的是写出object和embed标签外部的JavaScript文件，足以避免要求你的最终用户激活内容之前使用和相互影响。此外，上述解决办法正常运作，在所有的浏览器，因此，执行它确保最终用户的体验是一致的，无论其浏览器和平台组合。

Combining the technique described here along with the web player detection routine discussed in [Detecting the Unity Web Player using browser scripting](http://unity3d.com/support/documentation/Manual/Detecting%20the%20Unity%20Web%20Player%20using%20browser%20scripting.html) yields a predictable and optimized end-user experience. Here is an example of using both the detectUnityWebPlayer() and writeUnityTags() functions, where both functions are defined in an external JavaScript file named unityweb.js:

结合此处所述的技术随着网络播放器检测例行检测的Unity Web播放器使用浏览器脚本得到，讨论了预测和优化的最终用户体验。下面是一个同时使用detectUnityWebPlayer（）和writeUnityTags（）函数的例子，如果两个函数都在外部JavaScript文件名为unityweb.js定义：

<script type="text/javascript" language="javascript" src="unityweb.js"></script>

<script type="text/javascript" language="javascript">

// check for the Unity Web Player and respond appropriately

var tIsInstalled = detectUnityWebPlayer();

if (tIsInstalled) {

// call the writeUnityTags function

writeUnityTags("MyFile.unity3d", 640, 480);

} else {

// write out a simple message prompting the user to install the Unity Web Player

document.write("<div align='center'> \n");

document.write(" This content requires the Unity Web Player,");

document.write(" please use the link below to install the player today:<br /><br /> \n");

document.write(" <a href='http://www.unity3d.com/unity-web-player-2.x'> \n");

document.write(" Install the Unity Web Player \n");

document.write(" </a> \n");

document.write("</div> \n");

}

</script>

[**Unity Manual**](http://unity3d.com/support/documentation/Manual/index.html) > [**Advanced**](http://unity3d.com/support/documentation/Manual/Advanced.html) > [**Web Player Debugging**](http://unity3d.com/support/documentation/Manual/Web%20Player%20Debugging.html)

**Web Player Debugging**

There might be times during development when you need to obtain information from the logs of a webplayer you've built. Usually you need to see these files when your web player has experienced some problem, and you need to see where exactly the problem occurred.

**Mac OS X Log Location**

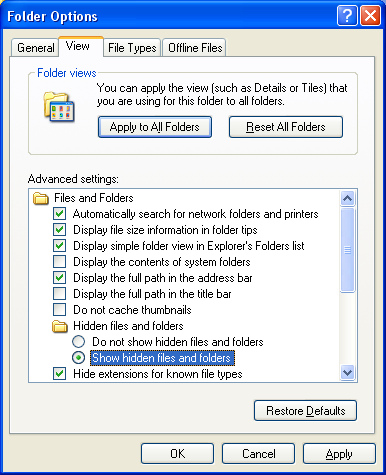
On Mac OS X, all web player log files are output to the console log: **~/Library/Logs/Unity/Player.log**. This log can be accessed via standard **Console.app** utility application, or from Unity editor menu **Help -> Open Player console log**.

**Windows Log Location**

On Windows, the web player log files are stored in the TEMP directory. This can vary slightly from computer to computer. Most commonly, the files are located in **C:\Documents and Settings\(current user)\Local Settings\temp\UnityWebPlayer\log**

**Please note:** The Local Settings folder is hidden by default. In order to see it, you must enable viewing of hidden folders from **Tools->Folder Options...->View (tab)**.

*Enabling viewing of hidden folders in Windows XP*



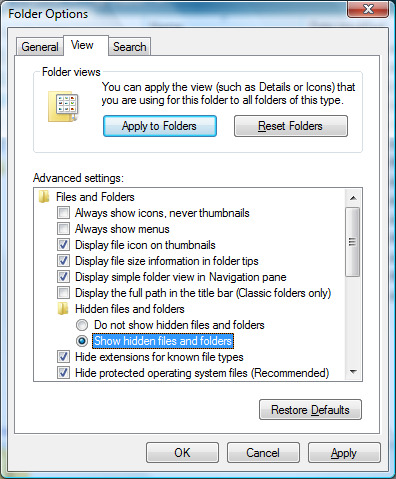
Inside this folder, you will see text files that begin with either "log\_" or "manager\_". Viewing these files will show you all the Debug.Log calls. You can safely delete the files inside this folder at any time.

**Windows Vista/IE7 Location**

If the user is running Vista and using IE7 **and** UAC is enabled, the log files will be located in a different place. This is due to the "enhanced" security of Windows Vista and IE7. That location is **C:\Users\(current user)\AppData\Local\Temp\Low\UnityWebPlayer\log**

**Please note:** The AppData folder is hidden by default. In order to see it, you must enable viewing of hidden folders from **Tools->Folder Options...->View (tab)**. The Tools menu is hidden by default, but can be displayed by pressing the Alt key once.

*Enabling viewing of hidden folders in Windows Vista*



[**Unity Manual**](http://unity3d.com/support/documentation/Manual/index.html) > [**Advanced**](http://unity3d.com/support/documentation/Manual/Advanced.html) > [**Plugins - Pro only feature**](http://unity3d.com/support/documentation/Manual/Plugins.html)

**Plugins - Pro only feature**

Unity has extensive support for C, C++ or Objective-C based **Plugins**. Plugins will work in standalones only. They are disabled when building a Web Player for security reasons.

In order to use a plugin you need to do two things:

* Write a plugin in a C based language and compile it.
* Create a C# script which calls functions in the plugin to do something.

So the plugin provides a simple C interface. The script then invokes the functions exposed by the plugin.

Here is a very simple example:

**The C file of a minimal plugin:**

float FooPluginFunction () { return 5.0F; }

**A C# script that uses the plugin:**

using UnityEngine;

using System.Runtime.InteropServices;

class SomeScript : MonoBehaviour

{

// This tells unity to look up the function FooPluginFunction inside the plugin named "PluginName"

[DllImport ("PluginName")]

private static extern float FooPluginFunction ();

void Awake ()

{

// Calls the FooPluginFunction inside the PluginName plugin

// And prints 5 to the console

print (FooPluginFunction ());

}

}

**Building a plugin for Mac OS X**

If you are building a plugin for Mac OS X, you have to create a bundle. The easiest way to do this is using XCode. Use **File->NewProject...** and select the Bundle - Carbon Bundle preset.

If you are using C++ (.cpp) or Objective-C (.mm) to implement the plugin you have to make sure the functions are declared with C linkage to avoid name mangling issues.

extern "C"

{

float FooPluginFunction ();

}

**Building a plugin for Windows**

Plugins on Windows are DLL files with exported functions. Practically any language or development environment that can create DLL files can be used to create plugins. Again, if you use C++, declare functions with C linkage to avoid name mangling issues.

**Using your plugin from C#**

Once you have built your bundle you have to copy it to **Assets->Plugins** folder. Unity will then find it by its name when you define a function like this:

[DllImport ("PluginName")]

private static extern float FooPluginFunction ();

Please note that **PluginName** should not include the extension of the filename.

**Deployment**

For cross platform plugins you have to include both .bundle (for Mac) and .dll (for Windows) files in Plugins folder. Once you have placed your plugins in the Plugins folder there is no more work required on your side. Unity automatically picks the right plugin for the right deployment platform and includes it with the player.

**Examples**

**Midi Plugin**

A complete example of the Plugin interface can be found [here](http://unity3d.com/tutorials/midiplugin.zip).

This is a complete MidiPlugin for OS X which uses Apple's CoreMidi API. It provides a simple C API and a C# class using the C API. The C# class contains a high level API, with easy access to NoteOn and NoteOff events and their velocity.

**Texture Plugin**

An example how to assign image data to a texture from C++ directly to OpenGL (note that is will only work when Unity is using OpenGL renderer). This example includes both XCode (for Mac) and Visual Studio (for Windows) project files. The plugin with accompanying Unity project can be found [here](http://unity3d.com/support/resources/example-projects/texture-plugins).

**More information**

[Mono Interop with native libraries](http://www.mono-project.com/Interop_with_Native_Libraries).

[P-invoke documentation on MSDN](http://msdn2.microsoft.com/en-us/library/fzhhdwae.aspx).

[**Unity Manual**](http://unity3d.com/support/documentation/Manual/index.html) > [**Advanced**](http://unity3d.com/support/documentation/Manual/Advanced.html) > [**Build Player Pipeline**](http://unity3d.com/support/documentation/Manual/Build%20Player%20Pipeline.html)

# Build Player Pipeline

When building a player, you sometimes want to modify the built player in some way. For example you might want to add a custom icon, copy some documentation next to the player or build an Installer. Doing this manually can become tedious and if you know how to write sh or perl scripts you can automate this task.

After building a player Unity automatically looks for a sh or perl script called **PostprocessBuildPlayer** (without any extension) in your Project's Assets->Editor folder. If the file is found, it is invoked when the player finishes building.

In this script you can post process the player in any way you like. For example build an installer out of the player.

You can use perl, sh or any other commandline compatible language.

Unity passes some useful command line arguments to the script, so you know what kind of player it is and where it is stored.

The current directory will be set to be the project folder, that is the folder containing the Assets folder.

#!/usr/bin/perl

my $installPath = $ARGV[0];

# The type of player built:

# "dashboard", "standaloneWin32", "standaloneOSXIntel", "standaloneOSXPPC", "standaloneOSXUniversal", "webplayer"

my $target = $ARGV[1];

# What optimizations are applied. At the moment either "" or "strip" when Strip debug symbols is selected.

my $optimization = $ARGV[2];

# The name of the company set in the project settings

my $companyName = $ARGV[3];

# The name of the product set in the project settings

my $productName = $ARGV[4];

# The default screen width of the player.

my $width = $ARGV[5];

# The default screen height of the player

my $height = $ARGV[6];

print ("\n\*\*\* Building at '$installPath' with target: $target \n");

In order to see this feature in action please visit the [Example Projects](http://unity3d.com/support/resources/assets/) page on our website and download the **PostprocessBuildPlayer** example package file and import it for use into your own project. It uses the Build Player Pipeline feature to offer customized post-processing of web player builds in order to demonstrate the types of custom build behavior you can implement in your own **PostprocessBuildPlayer** script.

[**Unity Manual**](http://unity3d.com/support/documentation/Manual/index.html) > [**Advanced**](http://unity3d.com/support/documentation/Manual/Advanced.html) > [**Command line arguments**](http://unity3d.com/support/documentation/Manual/Command%20Line%20Arguments.html)

# Command line arguments

It is possible to start Unity Editor with command line arguments to make it run certain tasks upon opening. This allows automated game builds, test suites and so on. See [below](http://unity3d.com/support/documentation/Manual/Command%20Line%20Arguments.html#editor#editor).

Additionally, standalone games built with Unity accept some command line parameters as well, see [below](http://unity3d.com/support/documentation/Manual/Command%20Line%20Arguments.html#player#player).

## Unity Editor command line arguments

### -batchmode

Run Unity in batch mode. This should always be used in conjunction with the other command line arguments as it ensures no pop up windows appear and eliminates the need for any human intervention. When an exception occurs during execution of script code, asset server updates fail or other operations fail Unity will immediately exit with return code 1.

### -quit

Quit Unity cleanly upon finishing execution of other command line arguments.

### -importPackage packagepath

Import the given [package](http://unity3d.com/support/documentation/Manual/HOWTO-exportpackage.html). No import dialog is shown.

### -createProject pathname

Create an empty project at the given path.

### -projectPath pathname

Open the project at the given path.

### -assetServerUpdate IP[:port] projectname username password

Force an update of the project in the [Asset Server](http://unity3d.com/support/documentation/Manual/Asset%20Server.html) given by IP:port. The port is optional and if not given it is assumed to be the standard one (10733). It is advisable to use this command in conjunction with the -projectPath argument to ensure you are working with the correct project. In no project name is given then the last project opened by Unity is used. If no project exists at the path given by -projectPath then one is created automatically.

### -executeMethod *ClassName.MethodName*

Execute the static method as soon as Unity is started and the project folder has been opened and after the optional asset server update has been performed. This can be used to do continous integration: perform Unit Tests, make builds, prepare some data. If you want to return an error from the commandline process you can either throw an exception which will cause Unity to exit with 1. Or call [EditorApplication.Exit](http://unity3d.com/support/documentation/ScriptReference/EditorApplication.Exit.html) with a non-zero code.

To use -executeMethod you need to have a script in an Editor folder and a static function in the class.

// C# example

using namespace UnityEditor

class MyEditorScript

{

static void PerformBuild ()

{

string[] scenes = { "Assets/MyScene.unity" };

BuildPipeline.BuildPlayer(scenes, ...);

}

}

// JavaScript example

static void PerformBuild ()

{

string[] scenes = { "Assets/MyScene.unity" };

BuildPipeline.BuildPlayer(scenes, ...);

}

### -nographics (Windows only)

When running in batch mode, do not initialize graphics device at all. This makes it possible to run your automated workflows on machines that don't even have a GPU.

### Example usage

Execute Unity in batch mode, execute MyEditorScript.MyMethod method, and quit upon completion.

C:\program files\Unity\Editor>Unity.exe -quit -batchmode -executeMethod MyEditorScript.MyMethod

Execute Unity in batch mode. Use the project path given and update from the asset server. Execute the given method after all assets have been downloaded and imported from the asset server. After the method has finished execution, automatically quit Unity.

/Applications/Unity/Unity.app/Contents/MacOS/Unity -batchmode -projectPath ~/UnityProjects/AutobuildProject -assetServerUpdate 192.168.1.1 MyGame AutobuildUser l33tpa33 -executeMethod MyEditorScript.PerformBuild -quit

## Unity Standalone Player command line arguments

Standalone players built with Unity also understand some command line arguments:

### -batchmode

Run the game in "headless" mode. The game will not display anything and won't accept user input. This is mostly useful for running servers of [networked games](http://unity3d.com/support/documentation/Components/Network%20Reference%20Guide.html).

### -force-opengl (Windows only)

Make the game use OpenGL for rendering, even if Direct3D is availabe. Normally Direct3D is used; and OpenGL is used only if Direct3D 9.0c is not available.

### -single-instance (Windows only)

Allow only one instance of the game to run at the time. If another game is already running, then launching it again with -single-instance will focus the existing one.

### -nolog (Windows only)

Do not produce output log. Normally output\_log.txt is written in the \*\_Data folder next to the game executable, where [Debug.Log](http://unity3d.com/support/documentation/ScriptReference/Debug.Log.html) output is printed.

### -force-d3d9-ref (Windows only)

Make the game run using Direct3D's "Reference" software renderer. [DirectX SDK](http://msdn.microsoft.com/en-us/directx/default.aspx) has to be installed for this to work. This is mostly useful for building automated test suites, where you want to ensure rendering is exactly the same no matter what's the graphics card.

[**Unity Manual**](http://unity3d.com/support/documentation/Manual/index.html) > [**Advanced**](http://unity3d.com/support/documentation/Manual/Advanced.html) > [**Shaders**](http://unity3d.com/support/documentation/Manual/Shaders.html)

**Shaders**

All rendering in Unity is done with *Shaders* - small scripts that let you configure the how the graphics hardware is set up for rendering. Unity ships with 40+ built-in shaders but you can extend this by making more yourself. The built-in shaders are documented in [Built-in Shader Guide](http://unity3d.com/support/documentation/Components/Built-in%20Shader%20Guide.html).

Shaders in Unity are written in a language called **ShaderLab**, which is similar to Microsoft's .FX files or NVIDIA's CgFX. If you want to write your own shaders, including custom vertex & fragment programs, this section and [ShaderLab reference](http://unity3d.com/support/documentation/Components/SL-Reference.html) will explain how.

To get you started from some examples, we ship the ShaderLab source code for all built-in shaders. You can get them [at Resources section](http://www.unity3d.com/support/resources/assets/built-in-shaders).

Read on for introduction to ShaderLab, and check out [ShaderLab reference](http://unity3d.com/support/documentation/Components/SL-Reference.html)!

* [Shaders: Getting started](http://unity3d.com/support/documentation/Manual/ShaderTut1.html)
* [Shaders: Vertex and Fragment Programs](http://unity3d.com/support/documentation/Manual/ShaderTut2.html)

[**Unity Manual**](http://unity3d.com/support/documentation/Manual/index.html) > [**Advanced**](http://unity3d.com/support/documentation/Manual/Advanced.html) > [**Shaders**](http://unity3d.com/support/documentation/Manual/Shaders.html) > [**Shaders: Getting started**](http://unity3d.com/support/documentation/Manual/ShaderTut1.html)

**Shaders: Getting started**

This tutorial will teach you how you can create your own shaders and make you game look a lot better!

Unity is equipped with a powerful shading and material language called **ShaderLab**. In style it is similar to CgFX and Direct3D Effects languages - it describes everything needed to display a **Material**, not just plain vertex/pixel shaders.

Shaders describe properties that are exposed in Unity's [Material Inspector](http://unity3d.com/support/documentation/Components/class-Material.html) and multiple shader implementations (**SubShaders**) targeted at different graphics hardware capabilities, each describing complete graphics hardware rendering state, fixed function pipeline setup or vertex/fragment programs to use. Vertex and fragment programs are written in high-level Cg programming language or low-level shader assembly.

In this tutorial we describe how to write shaders in ShaderLab using both fixed function and programmable pipelines. We assume that the reader has a basic understanding of [OpenGL](http://opengl.org/documentation/red_book) or Direct3D render states, fixed function and programmable pipelines and has some knowledge of [Cg](http://developer.nvidia.com/object/cg_toolkit.html), [HLSL](http://msdn.microsoft.com/en-us/library/bb509561%28VS.85%29.aspx) or [GLSL](http://www.opengl.org/documentation/glsl) programming languages. Some shader tutorials and documentation can be found on [NVIDIA](http://developer.nvidia.com/page/home.html) and [AMD](http://developer.amd.com/GPU/Pages/default.aspx) developer sites.

**Getting started**

To create a new shader, either choose **Assets->Create->Shader** from the menubar, or duplicate an existing shader, and work from that. The new shader can be edited by double-clicking it in the **Project View**.

We'll start with a very basic shader:

Shader "Tutorial/Basic" {

Properties {

\_Color ("Main Color", Color) = (1,0.5,0.5,1)

}

SubShader {

Pass {

Material {

Diffuse [\_Color]

}

Lighting On

}

}

}

This simple shader demonstrates one of the most basic shaders possible. It defines a color property called **Main Color** and assigns it a default value of rose-like color (red=100% green=50% blue=50% alpha=100%). It then renders the object by invoking a **Pass** and in that pass setting the diffuse material component to the property **\_Color** and turning on the vertex lighting.

To test this shader, create a new material, select the shader from the drop-down menu (**Tutorial->Basic**) and assign the Material to some object. Tweak the color in the Material Inspector and watch the changes. Time to move onto more complex things!

**Basic Vertex Lighting**

If you open an existing complex shader, it can be a bit hard to get a good overview. To get you started, we will dissect the built-in **VertexLit** shader that ships with Unity. This shader uses fixed function pipeline to do standard per-vertex lighting.

Shader "VertexLit" {

Properties {

\_Color ("Main Color", Color) = (1,1,1,0.5)

\_SpecColor ("Spec Color", Color) = (1,1,1,1)

\_Emission ("Emmisive Color", Color) = (0,0,0,0)

\_Shininess ("Shininess", Range (0.01, 1)) = 0.7

\_MainTex ("Base (RGB)", 2D) = "white" { }

}

SubShader {

Pass {

Material {

Diffuse [\_Color]

Ambient [\_Color]

Shininess [\_Shininess]

Specular [\_SpecColor]

Emission [\_Emission]

}

Lighting On

SeparateSpecular On

SetTexture [\_MainTex] {

constantColor [\_Color]

Combine texture \* primary DOUBLE, texture \* constant

}

}

}

}

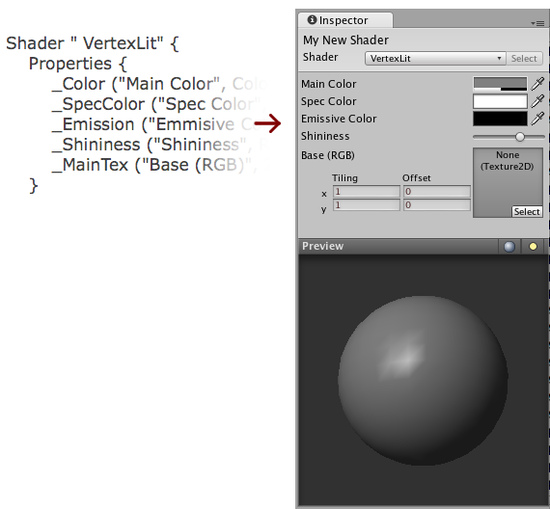
All shaders start with the keyword [**Shader**](http://unity3d.com/support/documentation/Components/SL-Shader.html) followed by a string that represents the name of the shader. This is the name that is shown in the **Inspector**. All code for this shader must be put within the curly braces after it: **{ }** (called a block).

* The name should be short and descriptive. It does not have to match the **.shader** file name.
* To put shaders in submenus in Unity, use slashes - e.g. **MyShaders/Test** would be shown as **Test** in a submenu called **MyShaders**, or **MyShaders->Test**.

The shader is composed of a **Properties** block followed by **SubShader** blocks. Each of these is described in sections below.

**Properties**

At the beginning of the shader block you can define any properties that artists can edit in the [Material Inspector](http://unity3d.com/support/documentation/Components/class-Material.html). In the *VertexLit* example the properties look like this:



The properties are listed on separate lines within the [**Properties**](http://unity3d.com/support/documentation/Components/SL-Properties.html) block. Each property starts with the internal name (**Color**, **MainTex**). After this in parentheses comes the name shown in the inspector and the type of the property. After that, the default value for this property is listed:



The list of possible types are in the [Properties Reference](http://unity3d.com/support/documentation/Components/SL-Properties.html). The default value depends on the property type. In the example of a color, the default value should be a four component vector.

We now have our properties defined, and are ready to start writing the actual shader.

**The Shader Body**

Before we move on, let's define the basic structure of a shader file.

Different graphic hardware has different capabilities. For example, some graphics cards support fragment programs and others don't; some can lay down four textures per pass while the others can do only two or one; etc. To allow you to make full use of whatever hardware your user has, a shader can contain multiple **SubShaders**. When Unity renders a shader, it will go over all subshaders and use the first one that the hardware supports.

Shader "Structure Example" {

Properties { /\* ...shader properties... }

SubShader {

// ...subshader that uses vertex/fragment programs...

}

SubShader {

// ...subshader that uses four textures per pass...

}

SubShader {

// ...subshader that uses two textures per pass...

}

SubShader {

// ...subshader that might look ugly but runs on anything :)

}

}

This system allows Unity to support all existing hardware and maximize the quality on each one. It does, however, result in some long shaders.

Inside each SubShader block you set the rendering state shared by all passes; and define rendering passes themselves. A complete list of available commands can be found in the [SubShader Reference](http://unity3d.com/support/documentation/Components/SL-SubShader.html).

**Passes**

Each subshader is a collection of passes. For each pass, the object geometry is rendered, so there must be at least one pass. Our VertexLit shader has just one pass:

// ...snip...

Pass {

Material {

Diffuse [\_Color]

Ambient [\_Color]

Shininess [\_Shininess]

Specular [\_SpecColor]

Emission [\_Emission]

}

Lighting On

SeparateSpecular On

SetTexture [\_MainTex] {

constantColor [\_Color]

Combine texture \* primary DOUBLE, texture \* constant

}

}

// ...snip...

Any commands defined in a pass configures the graphics hardware to render the geometry in a specific way.

In the example above we have a [**Material**](http://unity3d.com/support/documentation/Components/SL-Material.html) block that binds our property values to the fixed function lighting material settings. The command **Lighting On** turns on the standard vertex lighting, and **SeparateSpecular On** enables the use of a separate color for the specular highlight.

All of these command so far map very directly to the fixed function OpenGL/Direct3D hardware model. Consult [OpenGL red book](http://opengl.org/documentation/red_book) for more information on this.

The next command, [**SetTexture**](http://unity3d.com/support/documentation/Components/SL-SetTexture.html), is very important. These commands define the textures we want to use and how to mix, combine and apply them in our rendering. **SetTexture** command is followed by the property name of the texture we would like to use (**\_MainTex** here) This is followed by a **combiner block** that defines how the texture is applied. The commands in the combiner block are executed for each pixel that is rendered on screen.

Within this block we set a constant color value, namely the Color of the Material, **\_Color**. We'll use this constant color below.

In the next command we specify how to mix the texture with the color values. We do this with the **Combine** command that specifies how to blend the texture with another one or with a color. Generally it looks like this:

**Combine** **ColorPart**, **AlphaPart**

Here **ColorPart** and **AlphaPart** define blending of color (RGB) and alpha (A) components respectively. If **AlphaPart** is omitted, then it uses the same blending as **ColorPart**.

In our VertexLit example:

**Combine** texture \* primary DOUBLE**,** texture \* constant

Here **texture** is the color coming from the current texture (here **\_MainTex**). It is multiplied (\*) with the **primary** vertex color. Primary color is the vertex lighting color, calculated from the Material values above. Finally, the result is multiplied by two to increase lighting intensity (**DOUBLE**). The alpha value (after the comma) is **texture** multiplied by **constant** value (set with **constantColor** above). Another often used combiner mode is called **previous** (not used in this shader). This is the result of any previous **SetTexture** step, and can be used to combine several textures and/or colors with each other.

**Summary**

Our VertexLit shader configures standard vertex lighting and sets up the texture combiners so that the rendered lighting intensity is doubled.

We could put more passes into the shader, they would get rendered one after the other. For now, though, that is not nessesary as we have the desired effect. We only need one SubShader as we make no use of any advanced features - this particular shader will work on any graphics card that Unity supports.

The VertexLit shader is one of the most basic shaders that we can think of. We did not use any hardware specific operations, nor did we utilize any of the more special and cool commands that ShaderLab and Cg has to offer.

In the [next chapter](http://unity3d.com/support/documentation/Manual/ShaderTut2.html) we'll proceed by explaining how to write custom vertex & fragment programs using Cg language.

[**Unity Manual**](http://unity3d.com/support/documentation/Manual/index.html) > [**Advanced**](http://unity3d.com/support/documentation/Manual/Advanced.html) > [**Shaders**](http://unity3d.com/support/documentation/Manual/Shaders.html) > [**Shaders: Vertex and Fragment Programs**](http://unity3d.com/support/documentation/Manual/ShaderTut2.html)

**Shaders: Vertex and Fragment Programs**

This tutorial will teach you how to write custom vertex and fragment programs in Unity shaders. For a basic introduction to **ShaderLab** see the [Getting Started tutorial](http://unity3d.com/support/documentation/Manual/ShaderTut1.html).

Lets start with a small recap of the general structure of a shader:

Shader "MyShaderName" {

Properties {

// ... properties here ...

}

SubShader {

// ... subshader for graphics hardware A ...

Pass {

// ... pass commands ...

}

// ... more passes if needed ...

}

SubShader {

// ... subshader for graphics hardware B ...

}

// ... Optional fallback ...

FallBack "VertexLit"

}

Here at the end we introduce a new command:

**FallBack "VertexLit"**

The [**Fallback**](http://unity3d.com/support/documentation/Components/SL-Fallback.html) command can be used at the end of the shader; it tells which shader should be used if no **SubShaders** from the current shader can run on user's graphics hardware. The effect is the same as including all SubShaders from the fallback shader at the end. For example, if you were to write a bump-mapped shader, then instead of writing a very basic non-bump-mapped subshader for old graphics cards you can just fallback to built-in **VertexLit** shader.

The basic building blocks of the shader are introduced in the [first shader tutorial](http://unity3d.com/support/documentation/Manual/ShaderTut1.html) while the full documentation of [Properties](http://unity3d.com/support/documentation/Components/SL-Properties.html), [SubShaders](http://unity3d.com/support/documentation/Components/SL-SubShader.html) and [Passes](http://unity3d.com/support/documentation/Components/SL-Pass.html) are also available.

A quick way of building SubShaders is to use passes defined in other shaders. The command [**UsePass**](http://unity3d.com/support/documentation/Components/SL-UsePass.html) does just that, so you can reuse shader code in a neat fashion. As an example the following command uses the pass with the name "BASE" from the built-in **Specular** shader:

**UsePass "Specular/BASE"**

In order for **UsePass** to work, a name must be given to the pass one wishes to use. The [**Name**](http://unity3d.com/support/documentation/Components/SL-Name.html) command inside the pass gives it a name:

**Name "MyPassName"**

**Vertex and fragment programs**

We described a pass that used just a single texture combine instruction in the [first tutorial](http://unity3d.com/support/documentation/Manual/ShaderTut1.html). Now it is time to demonstrate how we can use vertex and fragment programs in our pass.

When you use vertex and fragment programs (the so called "programmable pipeline"), most of the hardcoded functionality ("fixed function pipeline") in the graphics hardware is switched off. For example, using a vertex program turns off standard 3D transformations, lighting and texture coordinate generation completely. Similarly, using a fragment program replaces any texture combine modes that would be defined in SetTexture commands; thus SetTexture commands are not needed.

Writing vertex/fragment programs requires a thorough knowledge of 3D transformations, lighting and coordinate spaces - because you have to rewrite the fixed functionality that is built into API's like OpenGL yourself. On the other hand, you can do much more than what's built in!

**Using Cg in ShaderLab**

Shaders in ShaderLab are usually written in [Cg programming language](http://developer.nvidia.com/page/cg_main.html) by embedding "Cg snippets" in the shader text. Cg snippets are compiled into low-level shader assembly by the Unity editor, and the final shader that is included in your game's data files only contains this low-level assembly. When you select a shader in the **Project View**, the **Inspector** shows shader text after Cg compilation, which might help as a debugging aid. Unity automatically compiles Cg snippets for both OpenGL and Direct3D, so your shaders that use Cg will just work on both. Note that because Cg code is compiled by the editor, you can't create Cg shaders from scripts at runtime.

In general, Cg snippets are placed inside Pass blocks. They look like this:

Pass {

*// ... the usual pass state setup ...*

**CGPROGRAM**

*// compilation directives for this snippet, e.g.:*

**#pragma vertex** vert

**#pragma fragment** frag

*// the Cg code itself*

**ENDCG**

*// ... the rest of pass setup ...*

}

The following example demonstrates a complete shader with Cg programs that renders object normals as colors:

Shader "Tutorial/Display Normals" {

SubShader {

Pass {

CGPROGRAM

#pragma vertex vert

#pragma fragment frag

#pragma fragmentoption ARB\_fog\_exp2

#include "UnityCG.cginc"

struct v2f {

V2F\_POS\_FOG;

float3 color : COLOR0;

};

v2f vert (appdata\_base v)

{

v2f o;

PositionFog( v.vertex, o.pos, o.fog );

o.color = v.normal \* 0.5 + 0.5;

return o;

}

half4 frag (v2f i) : COLOR

{

return half4( i.color, 1 );

}

ENDCG

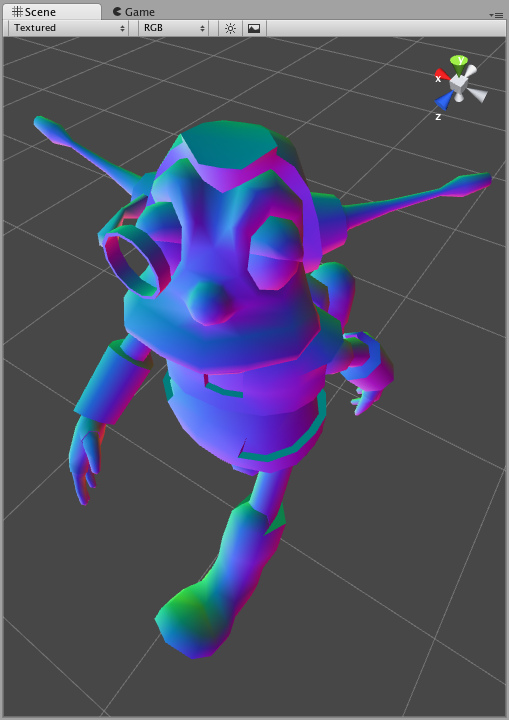
}

}

Fallback "VertexLit"

}

When applied on an object it will result in an image like this (if your graphics card supports vertex & fragment programs of course):



Our "Display Normals" shader does not have any properties, contains a single SubShader with a single Pass that is empty except for the Cg code. Finally, a fallback to the built-in **VertexLit** shader is defined. Let's dissect the Cg code part by part:

**CGPROGRAM**

**#pragma vertex** vert

**#pragma fragment** frag

**#pragma fragmentoption** ARB\_fog\_exp2

*// ... snip ...*

**ENDCG**

The whole Cg snippet is written between **CGPROGRAM** and **ENDCG** keywords. At the start compilation directives are given as **#pragma** statements:

* **#pragma vertex** **name** tells that the code contains a vertex program in the given function (**vert** here).
* **#pragma fragment** **name** tells that the code contains a fragment program in the given function (**frag** here).
* **#pragma fragmentoption** **name** adds an option to the compiled OpenGL fragment programs. Here we add support for exponential squared fog.

Following the compilation directives is just plain Cg code. We start by including a builtin Cg file:

#include **UnityCg.cginc**

The **UnityCg.cginc** file contains commonly used declarations and functions so that the shaders can be kept smaller. The file itself is found inside Unity application: **/Applications/Unity/Unity.app/Contents/CGIncludes/UnityCG.cginc**. Here we'll use **appdata\_base** structure, **V2F\_POS\_FOG** macro and **PositionFog** helper function from that file. We could just define them directly in the shader and not include the file of course.

Next we define a "vertex to fragment" structure (here named **v2f**) - what information is passed from the vertex to the fragment program. We pass the standard position and fog parameters and a **float3 color** parameter. The color will be computed in the vertex program and just output in the fragment program.

We proceed by defining the vertex program - **vert** function. Here we compute position and fog in the standard way (using helper function from UnityCG.cginc) and output input normal as a color:

o.color = v.normal \* 0.5 + 0.5;

Normal components are in -1..1 range, while colors are in 0..1 range, so we scale and bias the normal in the code above. Next we define a fragment program - **frag** function that just outputs the calculated color and 1 as the alpha component:

half4 frag (v2f i) : COLOR

{

return half4( i.color, 1 );

}

That's it, our shader is finished! Even this simple shader is very useful to visualize mesh normals.

Of course, this shader does not respond to lights at all, and that's where things get a bit more complicated; read on [Render Pipeline](http://unity3d.com/support/documentation/Components/SL-RenderPipeline.html) and [Light Attenuation](http://unity3d.com/support/documentation/Components/SL-Attenuation.html) pages in the reference for details.

**Using shader properties in Cg code**

When you define properties in the shader, you give them a name like **\_Color** or **\_MainTex**. To use them in Cg you just have to define a variable of a matching name and type. Unity will automatically set Cg variables that have names matching with shader properties.

Here is a complete shader that displays a texture modulated by a color. Of course, you could easily do the same in a texture combiner call, but the point here is just to show how to use properties in Cg:

Shader "Tutorial/Textured Colored" {

Properties {

\_Color ("Main Color", Color) = (1,1,1,0.5)

\_MainTex ("Texture", 2D) = "white" { }

}

SubShader {

Pass {

CGPROGRAM

#pragma vertex vert

#pragma fragment frag

#pragma fragmentoption ARB\_fog\_exp2

#include "UnityCG.cginc"

float4 \_Color;

sampler2D \_MainTex;

struct v2f {

V2F\_POS\_FOG;

float2 uv : TEXCOORD0;

};

v2f vert (appdata\_base v)

{

v2f o;

PositionFog( v.vertex, o.pos, o.fog );

o.uv = TRANSFORM\_UV(0);

return o;

}

half4 frag (v2f i) : COLOR

{

half4 texcol = tex2D( \_MainTex, i.uv );

return texcol \* \_Color;

}

ENDCG

}

}

Fallback "VertexLit"

}

The structure of this shader is the same as in the previous example. Here we define two properties, namely **\_Color** and **\_MainTex**. Inside Cg code we define corresponding variables:

float4 **\_Color**;

sampler2D **\_MainTex**;

See [Accessing Shader Properties in Cg](http://unity3d.com/support/documentation/Components/SL-PropertiesInPrograms.html) for more information.

The vertex and fragment programs here don't do anything fancy; vertex program uses the **TRANSFORM\_UV** macro from UnityCG.cginc to make sure texture scale&offset is applied correctly, and fragment program just samples the texture and multiplies by the color property.

Note that because we're writing our own fragment program here, we don't need any [SetTexture](http://unity3d.com/support/documentation/Components/SL-SetTexture.html) commands. How the textures are applied in the shader is entirely controlled by the fragment program.

**Summary**

We have shown how custom shader programs can be generated in a few easy steps. While the examples shown here are very simple, there's nothing preventing you to write arbitrarily complex shader programs! This can help you to take the full advantage of Unity and achieve optimal rendering results.

The complete ShaderLab reference manual is [here](http://unity3d.com/support/documentation/Components/SL-Reference.html). We also have a forum for shaders at [forum.unity3d.com](http://forum.unity3d.com) so go there to get help with your shaders! Happy programming, and enjoy the power of Unity and Shaderlab.

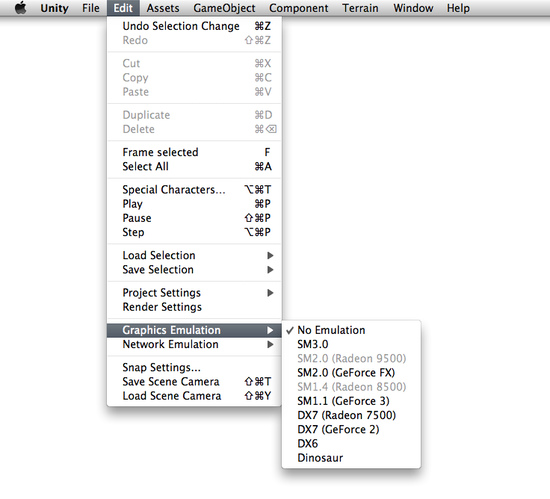
[**Unity Manual**](http://unity3d.com/support/documentation/Manual/index.html) > [**Advanced**](http://unity3d.com/support/documentation/Manual/Advanced.html) > [**Graphics Emulation**](http://unity3d.com/support/documentation/Manual/GraphicsEmulation.html)

**Graphics Emulation**

You can choose to emulate graphics hardware with less capabilities when working in Unity editor. This is very handy when writing custom shaders and rendering effects, and is a quick way to test how your game will look on that eight year old graphics card that someone might have.

To enable Graphics emulation, go to **Edit->Graphics Emulation**, and choose your desired emulation level.

*Enabling Graphics Emulation*



**Technical Details**

Graphics emulation limits the graphics *capabilities* that are supported, but it does not emulate the *performance* of graphics hardware. Your game in the editor will still be rendered by your graphics card; just more and more features will be disabled as you choose further emulation levels.

While emulation is a quick way to check out graphics capabilities, you should still test your game on actual hardware. This will reveal real performance, and any peculiarities of the specific graphics card, operating system or driver version.

**Emulation Levels**

Graphics emulation levels are the following:

**No Emulation**

No emulation is performed.

**SM3.0**

Emulates graphics card with Shader Model 3.0 level capabilities. Long vertex & fragment shader programs and support for built-in shadows.

**SM2.0 (Radeon 9500)**

Shader Model 2.0 capabilities, ATI Radeon style. Vertex & fragment programs, support for built-in shadows, ATI text fragment shader extension, 8 texture units.

**SM2.0 (GeForce FX)**

Shader Model 2.0 capabilities, NVIDIA GeForce style. Vertex & fragment programs, support for built-in shadows, 4 texture units.

**SM1.4 (Radeon 8500)**

Shader Model 1.4 capabilities. Vertex programs, ATI text fragment shader extension, 6 texture units. **Not supported**: fragment programs, built-in shadows.

**SM1.1 (GeForce 3)**

Shader Model 1.1 capabilities. Vertex programs, 4 texture units. **Not supported**: fragment programs, built-in shadows.

**DX7 (Radeon 7500)**

DirectX 7 level capabilities. Vertex programs (usually in software mode), 3 texture units. **Not supported**: fragment programs, built-in shadows.

**DX7 (GeForce 2)**

DirectX 7 level capabilities. Vertex programs (usually in software mode), 2 texture units, texture size limited to 2048, cubemap size limited to 512. **Not supported**: fragment programs, built-in shadows, volume textures, some complex texture combine modes.

**DX6**

DirectX 6 level capabilities. 2 texture units, texture size limited to 1024. **Not supported**: vertex & fragment programs, built-in shadows, cubemaps, volume textures, rectangle textures, three argument texture combine modes, DOT3 texture combine modes, projected textures.

**Dinosaur**

Emulates graphics cards everyone already forgot about! One texture unit, texture size limited to 512, vertex light count limited to 4. **Not supported**: vertex & fragment programs, built-in shadows, cubemaps, RenderTextures, volume textures, rectangle textures, three argument texture combine modes, DOT3 texture combine modes, projected textures.

When your graphics card does not support all capabilities of some emulation level, that level will be disabled. For example, Intel GMA950 (Intel 915/945/3000) card does not support Shader Model 3.0, so there's no way to emulate that level.

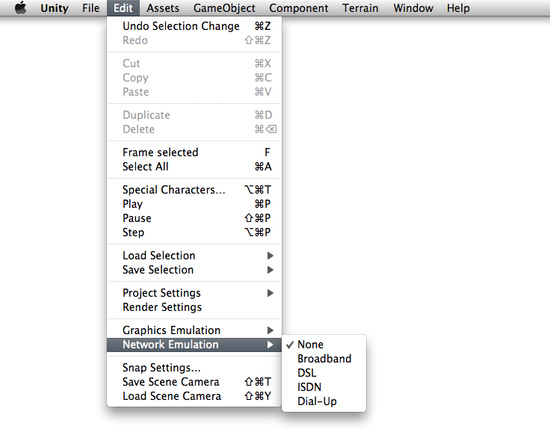
[**Unity Manual**](http://unity3d.com/support/documentation/Manual/index.html) > [**Advanced**](http://unity3d.com/support/documentation/Manual/Advanced.html) > [**Network Emulation**](http://unity3d.com/support/documentation/Manual/NetworkEmulation.html)

**Network Emulation**

As part of Unity's Networking feature set, you can choose to emulate slower internet connection speeds to test out your game experience for users in low-bandwidth areas.

To enable Network emulation, go to **Edit->Network Emulation**, and choose your desired connection speed emulation.

*Enabling Network Emulation*



**Technical Details**

Network emulation delays the sending of packets in networking traffic for the Network and NetworkView classes. The ping is artificially inflated for all options, the inflation value increasing as emulated connection speed gets slower. On the **Dial-Up** setting, packet dropping and variance is also introduced to simulate the worst possible connection ever. Emulation will persist whether you are serving the role of Server or Client.

Network emulation only affects the Network and NetworkView classes, and will not alter or emulate specialized networking code written using .NET sockets.

[**Unity Manual**](http://unity3d.com/support/documentation/Manual/index.html) > [**Advanced**](http://unity3d.com/support/documentation/Manual/Advanced.html) > [**Visual Studio C# Integration**](http://unity3d.com/support/documentation/Manual/VisualStudioIntegration.html)

**Visual Studio C# Integration**

Visual Studio C# Integration is a new feature in Unity 2.6

**What does this feature get me?**

A more sophisticated C# code writing environment.  
Think smart autocompletion, computer assisted changes to your codefiles, smart syntax highlighting and more.

**What's the difference between Express and Pro?**

VisualStudio C# 2008 is a product from Microsoft. It comes in an Express and a Profesional edition.  
The Express edition is free, and you can download it from here: <http://www.microsoft.com/express/vcsharp/>   
The Professional edition is not free, you can find out more information about it here: <http://www.microsoft.com/visualstudio/en-us/products/professional/default.mspx>

Unity's VisualStudio integration has two components:  
1) Unity creating and maintaining VisualStudio project files. **Works with Express and with Profesional.**  
2) Unity automatically opening VisualStudio when you doubleclick on a script, or error in Unity. **Works with Profesional only.**

**I got Visual Studio Express, how do I use it?**

* In Unity, select from the menu **Assets->Sync VisualStudio Project**
* Find the newly created .sln file in your Unity project (one folder up from your Assets folder)
* Open that file with Visual Studio Express.
* You can now edit all your script files, and switch back to Unity to use them.

**I got Visual Studio Profesional, how do I use it?**

* In Unity, go to Edit->Preferences, and make sure that Visual Studio is selected as your preferred external editor.
* Doubleclick a C# file in your project. Visual Studio should automatically open that file for you.
* You can edit the file, save, and switch back to Unity.

**Does this work with other versions than Visual Studio C# 2008?**

We've tested against Visual Studio 2008 and a prerelease version of Visual Studio 2010. At the time of this release, Visual Studio 2010 was not yet released, so we're not sure if Unity will work nicely with Visual Studio 2010 once it is released.

**A few things to watch out for:**

* Even though Visual Studio comes with its own C# compiler, and you can use it to check if you have errors in your c# scripts, Unity still uses its own C# compiler to compile your scripts. Using the Visual Studio compiler is still quite useful, because it means you don't have to switch to Unity all the time to check if you have any errors or not.
* Visual Studio's C# compiler has some more features than Unity's C# compiler currently has. This means that some code (especially newer c# features) will not give an error in Visual Studio but will give an error in Unity.
* Unity automatically creates and maintains a Visual Studio .sln and .csproj file. Whenever somebody adds/renames/moves/deletes a file from within Unity, Unity regenerates the .sln and .csproj files. You can add files to your solution from Visual Studio as well. Unity will then import those new files, and the next time Unity creates the project files again, it will create them with this new file included.
* Unity does not regenerate the Visual Studio project files after an AssetServer update, or a SVN update. You can manually ask Unity to regenerate the Visual Studio project files trough the menu: **Assets->Sync VisualStudio Project**

[**Unity Manual**](http://unity3d.com/support/documentation/Manual/index.html) > [**Advanced**](http://unity3d.com/support/documentation/Manual/Advanced.html) > [**Using External Version Control Systems with Unity**](http://unity3d.com/support/documentation/Manual/ExternalVersionControlSystemSupport.html)

**Using External Version Control Systems with Unity**

Unity offers an [Asset Server](http://unity3d.com/support/documentation/Manual/Asset%20Server.html) add-on product for easy integrated versioning of your projects. If you for some reason are not able use the Unity Asset Server, it is possible to store your project in any other version control system, such as Subversion, Perforce or Bazaar, although this requires some manual initial setup of your project and moving and renaming of assets has to be performed using your version control client and not inside Unity.

External Version Control is a Unity Pro feature.

Before checking your project in, you have to tell Unity to modify the project structure slightly to make it compatible with storing assets in an external version control system. This is done by selecting **Edit->Project Settings->Editor** in the application menu and enabling External Version Control support by clicking the **Enable** button. This will create a text file for every asset in the Assets directory containing the necessary bookkeeping information required by Unity. The files will have a .meta file extension with the first part being the full file name of the asset it is associated with. When moving or renaming assets in the version control system, make sure you also move or rename the .meta file accordingly.

When checking the project into a version control system, you should at least add the Assets directory to the system. If you want to track project and build settings as well you can also add Library/\*.asset, and Library/BuildPlayer.prefs. Do not add any other files or directories located inside the Library directory. When creating new assets, make sure both the asset itself and the associated .meta file is added to version control.

**Example: Creating a new project and importing it to a Subversion repository.**

First, let's assume that we have a subversion repository at svn://my.svn.server.com/ and want to create a project at svn://my.svn.server.com/MyUnityProject. Then follow these steps to create the initial import in the system:

1. Create a new project inside Unity and call it InitialUnityProject. You can add any initial assets here or add them later on.
2. Enable **Meta files** in **Edit->Project Settings->Editor**
3. Quit Unity
4. Delete the contents of the Library directory inside your project folder. Don't delete the directory itself.
5. Import the project folder into Subversion. If you are using the command line client, this is done like this from the directory where your initial project is located:  
   svn import -m"Initial project import" InitialUnityProject svn://my.svn.server.com/MyUnityProject  
   If successful, the project should now be imported into subversion and you can delete the InitialUnityProject directory if you wish.
6. Check out the project back from subversion  
   svn co svn://my.svn.server.com/MyUnityProject
7. Optional: Set up an ignore filter for the unversioned files inside the Library directory:  
   svn propedit svn:ignore MyUnityProject/Library  
   Subversion will open a text editor. Add the following file names to the file and save it.
   * AssetServerCacheV3
   * FailedAssetImports.txt
   * ScriptAssemblies
   * ScriptMapper
   * Temp
   * assetDatabase3
   * cache
   * guidmapper
   * metadata
   * unity default resources
   * unity editor resources
8. Open the checked out project with Unity by launching it while holding down the **Option** or the left **Alt** key. Opening the project will recreate the missing files deleted from the Library directory in step 4 above.
9. Add the project settings to the subversion repository:  
   svn add MyUnityProject/Library/\*.asset MyUnityProject/Library/BuildPlayer.prefs
10. Finally commit the changes. The project should now be set up and ready:  
    svn ci -m"Finishing project import" MyUnityProject