Autodesk®
Scaleform®

Unity Scaleform 集成概述

本文介绍 Unity-Scaleform 集成的主要特性。

作者： Ankur Mohan
版本： 1.03
上次编辑：2012 年 12 月 3 日

Autodesk®
GAMEWARE

Copyright Notice

Autodesk® Scaleform® 4.2

How to Contact Autodesk Scaleform:

Document	Unity-Scaleform Integration Overview
Address	Scaleform Corporation 6305 Ivy Lane, Suite 310 Greenbelt, MD 20770, USA
Website	www.scaleform.com
Email	info@scaleform.com
Direct	(301) 446-3200
Fax	(301) 446-3199

目录

1 引言 1

2 安装 2

3 发行版 3

4 演示 4

4.1 HelloWorldDemo 4

4.2 StarshipDown Demo 6

5 当前实现中的局限性 8

6 架构 8

7 使用集成 10

8 关键考虑因素 11

8.1 多线程架构 11

8.1.1 使用命名空间 11

8.1.2 创建和销毁 Scaleform 运行时 12

8.1.3 推进/显示 12

8.1.4 点击测试 13

8.1.5 确定当前视窗 13

8.1.6 从 ActionScript 调用 C# 函数 13

8.1.7 事件处理 14

8.1.8 在脚本中表示电影 14

8.1.9 寿命问题 14

8.1.10 Direct Access API 15

8.1.11 跟踪语句 15

8.1.12 部署 15

9 iOS 上的特殊考虑因素 16

9.1.1 的行为 16

10 渲染到纹理 17

10.1 在 Unity 集成中使用渲染纹理 18
10.2 命中测试 19
10.3 将焦点转移到 RTT 电影 20
10.4 添加阿尔法混合 20
10.4.1 透明电影 20
11 附录 22
11.1 Interop 22
11.2 在 Windows 上构建 23
11.2.1 在不调试的情况下运行演示 23
11.2.2 使用 Visual Studio 进行调试 24
11.3 在 iOS 上构建 25
11.4 在 Android 上构建 31
11.4.1 运行 HelloWorldDemo 32
11.4.2 创建自己的应用程序 32
11.4.3 使用 cygwin 在 Android 设备上启动 .apk 33

1 引言

本文介绍 Unity-Scaleform 集成的主要特性。我们假设用户初步了解采用 Adobe Flash（电影剪辑、事件等）进行的 UI 设计，并且对使用 Scaleform GFx（Advance/Display/ExternalInterface 等）有所熟悉。如果您不熟悉 Flash 和/或 Scaleform，请参阅我们的评估套件中包含的《Scaleform 4.2 入门》(Getting Started with Scaleform 4.2) 。

本文其余部分的内容如下：

- “安装”介绍安装程序包后需要执行什么操作。
- “发行版”(Distribution) 介绍此程序包中包含什么内容。
- “演示”介绍随此程序包分发的 HelloworldDemo 和 StarshipDown 演示。请注意，附录中提供有在 PC/iOS/Android 上构建这些演示的详细说明。
- “局限”概述此集成的当前版本的局限性。
- “架构”概述此集成的工作原理，以及有关托管代码 (Managed Code) 与非托管代码 (Unmanaged Code) 之间的接口连接的一些基本情况。
- “使用此集成”详细介绍在您自己的应用程序中使用此集成时所需执行的步骤。
- “关键考虑因素”概述使用此集成时需要注意的一些重要考虑因素。
- 最后一节概述与此产品相关的 Windows、iOS 和 Android 之间的区别。这一节对 Android/iOS 开发者来说尤其重要，因为本文绝大部分内容是从 Windows 用户的角度编写的。

附录中提供了有关 Windows、iOS 和 Android 平台的详细构建说明。

2 安装

如果在安装过程中选择默认值，此集成就应安装在 Documents\Autodesk\Scaleform\SF4.2_Unity 中。本文中的任何相关路径均引用此根目录。

安装后的第一步是导入 Integrations\Unity\HelloworldDemo 和 Integrations\Unity\StarshipDown 文件夹中封装的素材 (Asset)。为此，您可以双击 HelloworldDemo.unitypackage/StarshipDown.unitypackage 文件，也可以创建新的 Unity 项目并导入程序包。对于 HelloworldDemo，我们建议将素材导入 HelloworldDemo 文件夹，这样，导入完成之后，您就会拥有下面的目录结构：

Integrations\Unity\HelloworldDemo\Assets\
Integrations\Unity\HelloworldDemo\Library\
等等。

这样一来，您的文件夹结构就与本文其余部分显示的文件夹结构相一致。

3 发行版

此发行版包括下列内容：

Doc：包含一个全面的说明文档，其内容涵盖我们 SDK 的各个方面。如果您不熟悉 Scaleform，就最好从位于 Doc/GFx 的“入门”(Getting Started) 指南开始。您还应熟悉我们的分析工具 – AMP（内存及性能分析器）。

Bin：

- Win32 或 MacOS：用于 PC/Mac 的水印版播放器可执行程序 (GFxMediaPlayer.exe)。您可以使用此可执行程序在开发平台 (PC/Mac) 上运行 Flash 文件，然后再将这些文件与 Unity 一起使用。一般情况下，只能将 Flash 文件拖放到播放器窗口上。请注意，此发行版不包含可在移动设备上启动的播放器可执行程序。
- Data/AS3：可用于测试用途的一些示例 SWF/FLA 文件。
- AmpClient.exe 是我们的分析器工具，可用来获取有关 Unity 编辑器中或移动设备上运行的 Scaleform 内容的详细的帧级渲染和内存统计数据。Exporter.exe 是导出应用程序，用来从 swf 文件创建压缩的 gfx 文件。有关这些应用程序的更多信息，请参阅说明文档。

Lib：包含在 iOS 上构建您的应用程序所必需的水印 Scaleform 库。PC/Android 上不需要

资源：包含我们的 CLIK（通用轻便接口工具箱）库，该库包含常用 UI 组件（如按钮、滚动列表等）的预装实现。有关 CLIK 入门的更多信息，请参阅我们的说明文档。此发行版包含的演示广泛使用 CLIK，我们鼓励您熟练使用 CLIK，因为这会使 UI 开发更加方便快捷。

Integrations/Unity/Doc：此文档的主机文件夹

Integrations/Unity/Bin：包含 Scaleform 二进制文件的 Debug/Release/Shipping 版本。这些二进制文件在 PC 上是动态链接库 (dll)，而在 Android 上是共享对象 (.so)。对于 iOS，请参阅下面的“Integrations/Unity/Lib”。将这些二进制文件的发布版本也复制到特定项目的文件夹（例如，HelloworldDemo/Assets/Plugins/ for Mac and PC 和 HelloworldDemo/Assets/Plugins/Android for Android）之中。在编辑器中按“播放”(Play) 或以独立模式运行您的应用程序时，Unity 就会自动加载动态库。

Integrations/Unity/Lib：包含 iOS 上的 GFxUnity3DInternal 的 Debug/Release/Shipping 版本。此 lib 是用于 Unity 与 Scaleform 之间的接口的一个包装器 (Wrapper)，用 XCode 构建您的 iOS 应用程序时，就需要将此 lib 与核心 Scaleform lib 链接在一起。

由于 PC/Android 使用动态链接库，此文件夹在那些平台上是空的。

Integrations/Unity/Src：包含 SFExports 以及在 iOS 上构建您的应用程序所需的一些其它文件。PC/Android 上不需要。

Integrations/Unity/

HelloWorldDemo

StarshipDown:我们的演示项目，支持所有平台。

现在，我们就介绍一下在演示项目（以 HelloworldDemo 为例）中分发 Scaleform 素材的情况。请注意，您必须首先导入随此发行版一起提供的 HellworldDemo/StarshipDown.unitypackage 文件。

每个项目使用的 Flash 素材（SWF/FLA 文件）均位于 HelloWorldDemo\Assets\StreamingAssets。部署时，Unity 自动将此文件夹中的文件复制到目标平台上。

此集成中的 C# 脚本位于 HelloworldDemo\Assets\Plugins\SF 和 HelloworldDemo\Assets\Scripts\Scaleform 目录。第一个文件夹包含实现该插件的核心功能的脚本。这些脚本在 StarshipDown 与 HelloworldDemo 之间通用。如果创建一个使用 Scaleform 的新 Unity 应用程序，您就只能将这些脚本复制到一个相应的位置。第二个文件夹包含针对应用程序的代码的脚本。

Scaleform 二进制文件在 Android 和 PC 上分别位于 HelloworldDemo\Assets\Plugins\Android 和 HelloworldDemo\Assets\Plugins\PC。如前所述，此发行版包含位于 Unity\bin 的这些二进制文件的 Debug/Release/Shipping 版本。为了方便起见，也将这些二进制文件的发布版本复制到每个演示的 Plugins 文件夹中。

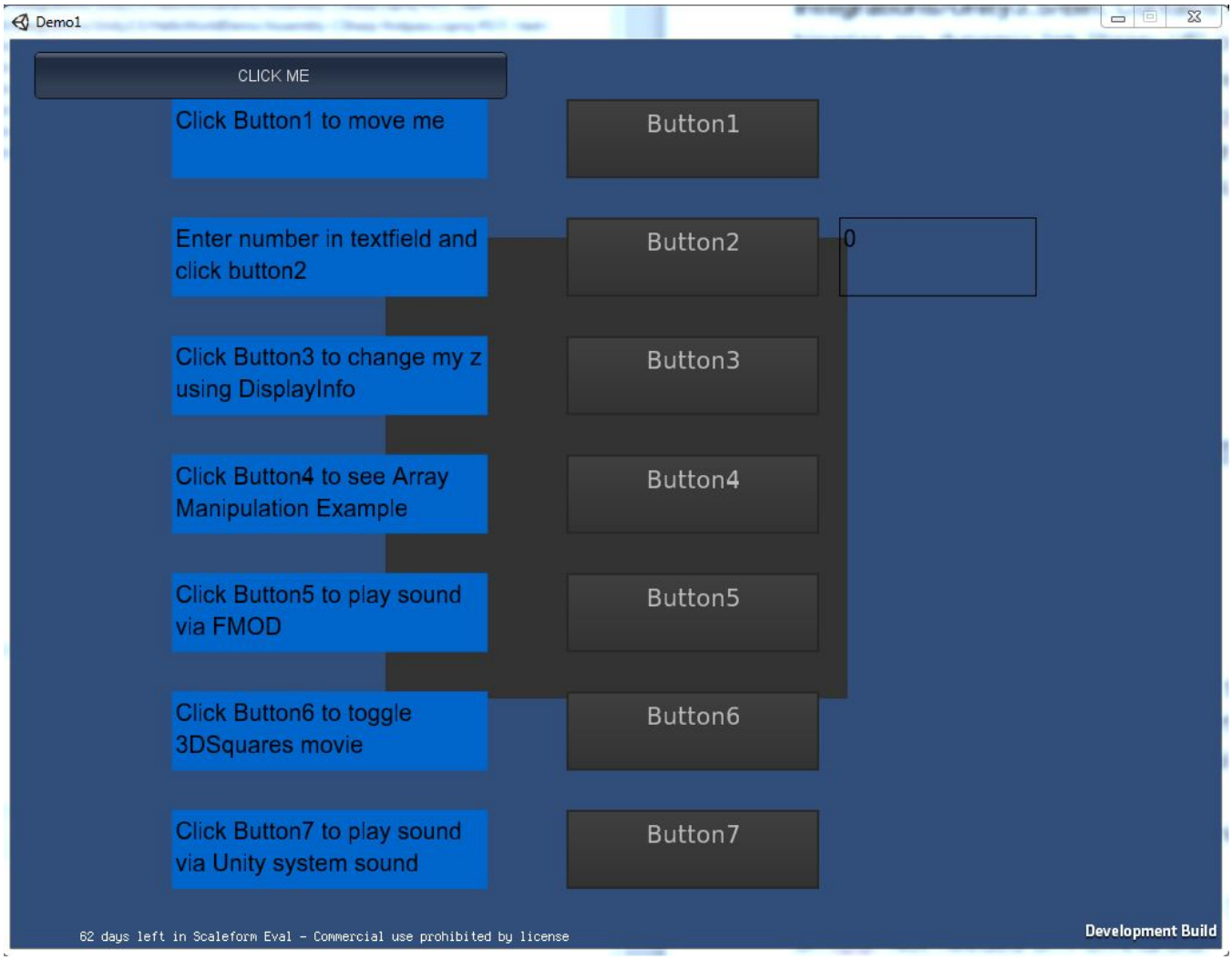
在 iOS 上， 您将使用位于每个演示的 iOS_Project 文件夹的 XCode 项目来构建您的应用程序。此项目链接 Scaleform lib 以及构建您的 iOS 应用程序所需的所有其它 Unity 和系统 lib。

尽管 Unity 支持多种脚本语言，迄今为止我们还是用 C# 实现我们的集成。我们认为 C# 对于 Unity 来说是使用最广泛的脚本语言。如果想在您的脚本编写中使用 JS/Boo，把 C# 逻辑转换为其它语言应该是非常简单的。

4 演示

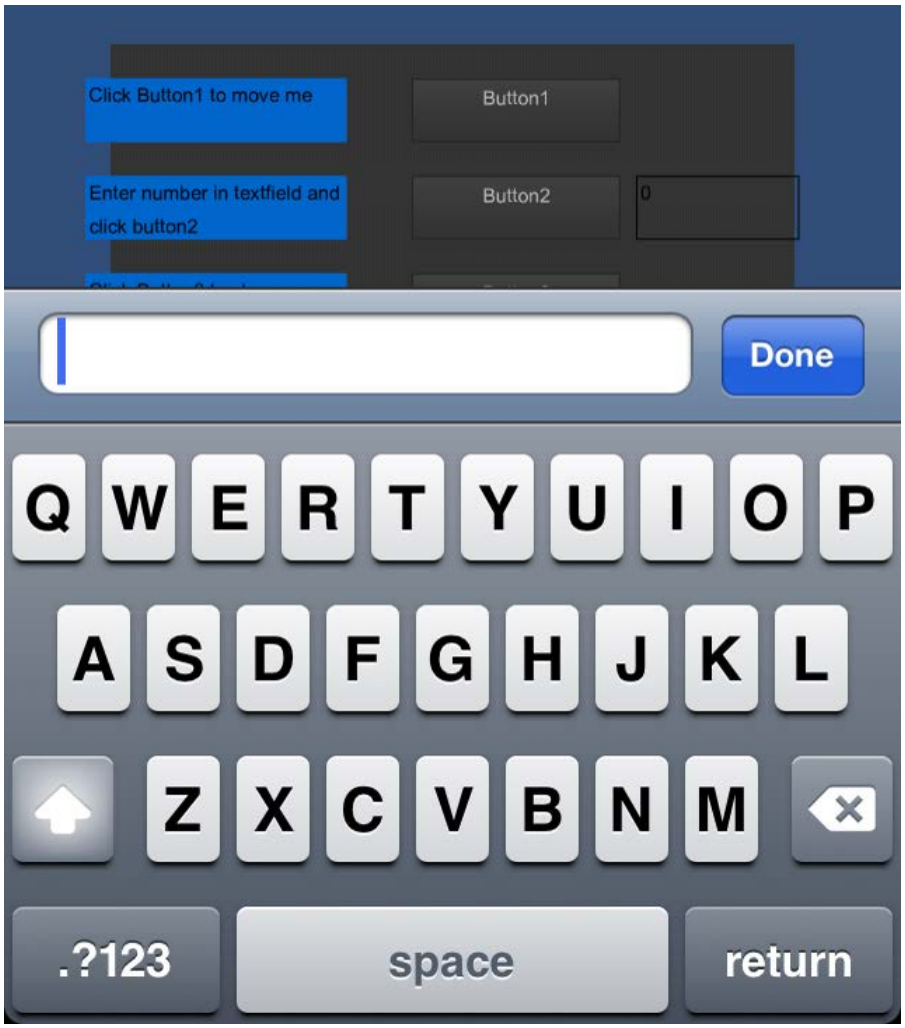
4.1 HelloWorldDemo

此简单但有用的演示显示使用 Scaleform 的许多重要方面。此演示的代码包含在 Demo1.fla、MyCamera.cs 和 UI_Scene_Demo1.cs 中。



1. **使用 Direct Access API 修改 Flash 对象的属性：**单击按钮 1 可向左/右移动矩形，而单击按钮 3 则可更改矩形的 z 值。这还可以演示 3Di，即将一个 z 值指定给 Flash 电影剪辑的能力。

2. **在不同平台上输入文本：**单击文本字段，并键入一个数字。现在，当您单击按钮 2 时，后台中的立方体将以一个角速度旋转，该角速度与文本字段中键入的数字相一致。在移动平台上，当该文本字段收到焦点时，会弹出一个虚拟键盘，您可以键入数字。



3. **在 Actionscript 与 C# 之间来回传递复杂数据：**当您点击按钮 4 时，就会将一组结构（包含名称和核心）传递到 C#，在那里可以对其进行修改。这将表明实现记分牌/排行榜所需的管道。

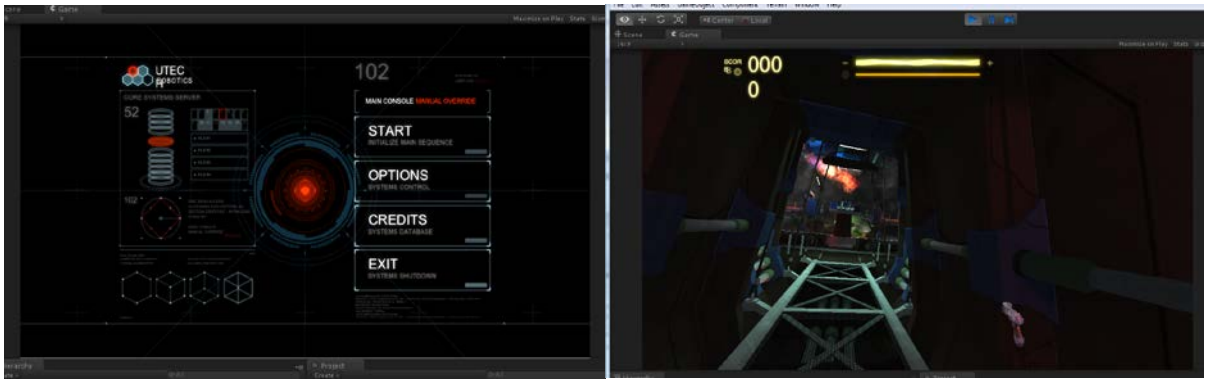
4. **播放声音 via FMOD：**点击按钮 5，就会听到 electric.wav，这是嵌入 demo1.swf 的一个波形文件 (Wave File)。此演示使用 FMOD 播放声音，并且目前仅支持 PC。

5. 通过 Unity 系统声音播放声音：当您单击按钮 7 时，就会听到 “AutoTurretFire.wav” – 一个外部波形文件。此演示使用 Unity 的本机系统声音播放支持。

6. **用 C# 创建和销毁电影：**点击按钮 6 可切换 3Dsquares 电影。

4.2 StarshipDown Demo

StarshipDown 演示用图说明制作一些常用 UI 屏幕（如主菜单、HUD 和暂停菜单），并显示 HUD 上的简单游戏的结果。StarShipDemo 从一个主菜单 (Main Menu) 屏幕开始。一按 START（开始），就会异步加载游戏关卡。加载游戏关卡时，会显示一个加载屏。当 StarshipDown 关卡完成加载时，会显示一个得分和生命值/弹药（Health/Ammo）栏，作为 HUD 的组成部分。得分/生命值数据在您演练该关卡和射击恶魔时会发生变化。按 Escape，就会显示一个 PAUSE（暂停）菜单。



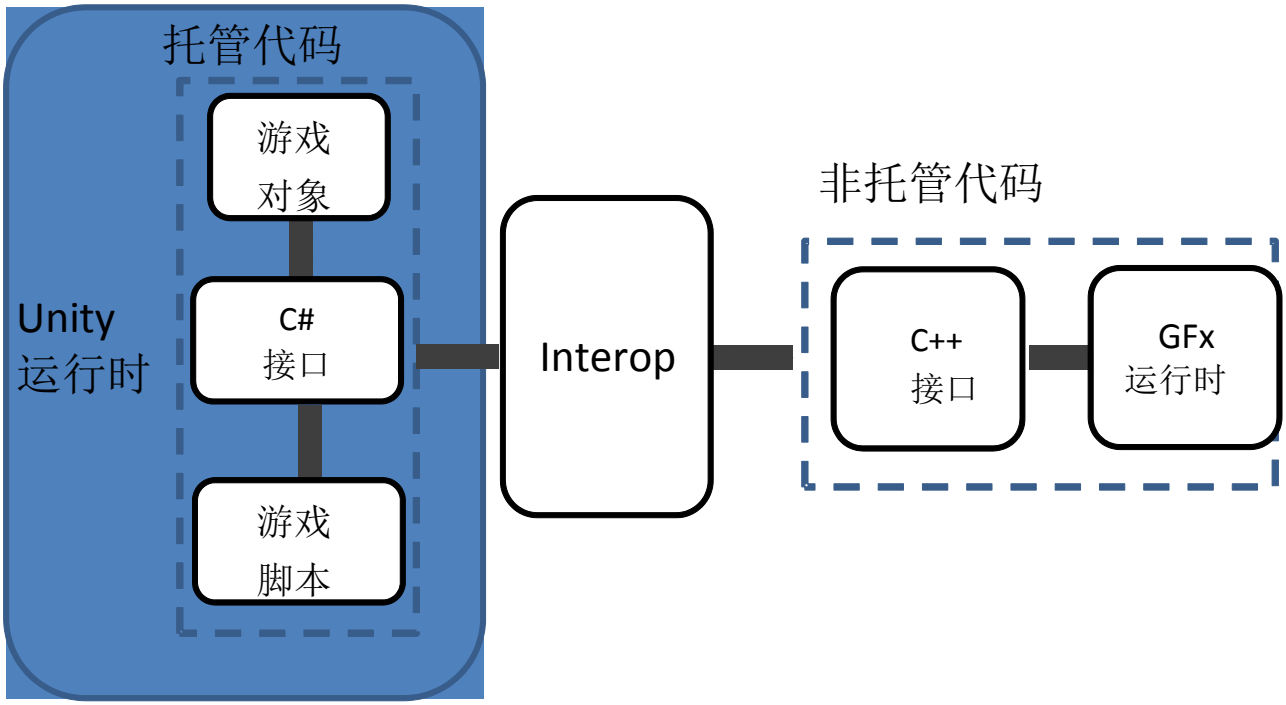
5 当前实现中的局限性

目前，仅支持渲染到主帧缓冲区。尚未实现渲染到纹理。也不支持多层渲染。在此集成的未来版本中将会实现这些功能。

6 架构

在 Windows 上，此集成包含一个 dll，它包装 Scaleform 运行时，并导出可从脚本层调用的一系列函数。在 iOS 上，Scaleform 库链接到此应用程序中。Android 与 Windows 相似，它将 Scaleform 运行时包含在一个共享对象文件中。

尽管 Unity 支持许多脚本语言（C#、JS、
Boo），我们还是仅提供脚本层的 C# 实现。下面的图表显示集成结构。



此集成有两个主要组件。

- C# 层（托管代码）：包含 SFManager 类的 C# 实现。此层负责实例化 C++ 层和 Scaleform 运行时、调用 Advance/Display、将鼠标/按键事件传递到 Scaleform、从 Scaleform 接收回调，并与游戏中的各种脚本通信。

- C++ 层（非托管代码）：此层充当 Scaleform API 的包装器，并执行与 C# 层通信所需的一些封送 (Marshaling)。

附录中的 “Interop” 一节将会更加详细地介绍托管代码与非托管代码之间的通信。

7 使用集成

本节列举在 Unity 中使用 Scaleform 时需要进行的关键步骤。

第一步是创建一个游戏对象，并将一个脚本附加到它上面，该游戏对象初始化 Scaleform 运行时并实例化 Scaleform 管理器 (Scaleform Manager)。“Main Camera” 游戏对象在 StarshipDown 演示中执行此角色。将 “ScaleformCamera” 脚本附加到此对象。Scaleform Camera 与其基类 SFCamera 一起执行初始化 Scaleform 并与 Unity 事件系统挂钩所需的所有步骤。

SFCamera 公开一个 InitParams 结构，该结构可在编辑器中检查，并且可用来设置各种 Scaleform 状态。例如，您可以设置是否初始化 Video/Sound（视频/声音）子系统、使用 ActionScript（AS2/3/二者兼有）的哪个版本，等等。

第二步是通过实例化 Movie（电影）派生的类来创建 Scaleform 电影。Movie 已经包含执行与电影相关的各种任务（如创建/销毁电影、通过与 Scaleform 插件互动来调用“推进”(Advance)/“显示”(display)）所必需的核心功能。

第三步是在您的 Movie 派生的类中添加自定义事件处理程序。通过在 C# 中使用反射，此集成使得添加事件处理程序非常容易。您只需添加具有与相应的 externalInterface 回调相同的名称并接受相同的参数的函数。StarshipDown 演示广泛利用此模式。例如，查看 UI_Scene_HUD.cs 文件。

在下一节中，我们重点介绍在使用此集成时应了解的许多关键考虑因素。

8 关键考虑因素

8.1 多线程架构

Unity 引入一个多线程渲染器，因而渲染和游戏更新在不同线程中发生。此架构与 Scaleform 4.1 巧妙地配合在一起，而 Scaleform 4.1 对于“推进”和“显示”也使用不同的线程。在多线程架构中，必须在渲染线程中创建所有与渲染相关的资源（如 HAL）。由于通常通过从游戏脚本（在主游戏线程上运行）调用导出的函数来驱动插件，因而不能从脚本发出任何与渲染相关的调用。为了实现让像我们的执行渲染的插件这样的低级插件正常工作，Unity 引入某种新的 API，这种 API 从渲染线程调用一个导出的函数，该函数具有一个预定义的名称和签名。我们使用此 API 同时支持 D3Dx 以及多线程渲染。

有关此话题的更多信息，请参阅 Unity 说明文档：
Unity Manual（Unity 手册）> Advanced（高级）> Plugins (Pro/Mobile-Only Feature)（插件（仅限 Pro/Mobile 特性）> Low-level Native Plugin Interface（低级本机插件接口）

另请参阅：
`void UnitySetGraphicsDevice (void* pdevice, int deviceType, int eventType)`
在 SFExports.cpp（此文件仅随 iOS 发行版一起分发）中，以及 SFCamera 中的
`StartCoroutine("CallPluginAtEndOfFrames")` 和 `GL.IssuePluginEvent(0)`

GL.IssuePluginEvent 方法可用来把命令放到 Unity 的渲染队列上。这些命令将在 Unity 的渲染线程上执行。为了设置各种渲染参数（如 GlyphCacheParams），我们还保留我们自己的共享队列，并在从 Unity 内调用“显示”(Display) 时处理该共享队列。此队列使用 Platform（平台）项目（标准 Scaleform SDK 的一部分）中提供的实现方法。

另外，如 Unity 说明文档所述，渲染仅在 PC（而非 iOS/Android）上具有多线程特性。此集成同时支持多线程渲染和单线程渲染。

8.1.1 使用命名空间

所有与 Scaleform 集成相关的 C# 代码均包含在 Scaleform 或 Scaleform::GFx 命名空间中。这与 Scaleform SDK 中使用的命名空间和类命名约定相一致。例如，Movie（电影）类就是同时用 C++ 和 C# 在 Scaleform::GFx 命名空间中定义的。Scaleform 命名空间中包含 C++ 中没有副本的 C# 类（如 SFManager）。这有助于避免与可能具有相同名称的 Unity 类发生冲突。

8.1.2 创建和销毁 Scaleform 运行时

在我们的集成中，有三个初始化步骤。首先是创建 Scaleform 运行时、FManager、Scaleform 渲染器和硬件抽象层 (HAL)。

在 PC 上初始化 Scaleform 运行时的操作是在 UnitySetGraphicsDevice（从 Unity 内调用）中完成的。假如 D3D 渲染器正在使用当中，在此函数中，也会将指向 D3D 设备的指针作为一个参数进行传递。在 iOS/Android 上，在 SF_Init 中执行 Scaleform 运行时初始化。

第二个步骤是在 Scaleform 加载程序 (Scaleform Loader) 上设置各种状态以及将 InitParams 结构 (SFCamera.cs) 中指定的设置传递到 Scaleform 运行时，这样，就可按照用户指定的设置初始化运行时。此操作是在 SF_Init 函数（从 SFCamera:Start 中的 Script 那里调用）完成的。

第三个步骤是初始化第一步中创建的 HAL 对象。在多线程系统中，HAL 初始化必须在渲染器线程上发生。这是通过在 Windows 上发出一个 GL.IssuePluginEvent 调用和在 iOS/Android 上发出一个 UnityRenderEvent 调用（eventId = 0) 来完成的。

SFManager 类包含该插件的 C# 部分的大部分实现。必须在创建任何电影之前初始化此类。

可用两种方法创建电影：通过调用 SFManager:CreateMovie，或者通过创建某个电影派生的类的新实例。两种方法都将一个 SFMovieCreationParams 对象作为封装电影名称、视窗参数等的一个参数。一种典型的电影创建调用如下所示：

```
demo1 = new UI_Scene_Demo1(SFMgr, CreateMovieCreationParams("Demo1.swf"));
```

如下所述，电影是通过其 ID 标识的，而该 ID 只是与该电影对应的 C++ Movie 指针的整数表示。SFManager 在内部维护一个电影列表，用来处理事件、调用 Advance/Display（推进/显示）等。

8.1.3 推进/显示

Flash 动画是在 Movie:Advance 期间推进的。在此集成中，C++ Advance 是在 SFManager.Advance 期间被调用的，而后者又是在 SFCamera.Update 期间被调用 。电影是在 Display（显示）期间调用的，而且，在 PC 上，由 Unity 通过 UnityRenderEvent 函数在渲染线程上调用 Display，而在 iOS/Android 上，则是通过调用 UnityRenderEvent（eventId = 1) 完成的。

Advance（推进）将上次更新以来过去的时间当作一个参数。这可用来控制推进电影的速度。假如您想要更加精细地控制某个电影的推进速度，可以覆盖 Movie 派生的类中的 Advance 方法。

在多数常用使用情况下，Advance 的默认实现就足够了。任何游戏电影通信（如 invoke per tick）均应通过覆盖 Movie.Update 函数（在 Advance 之前调用）进行实现。这使得游戏更新与电影更新恰当地平行进行。

8.1.4 点击测试

点击测试（Hit-Testing）用来检查某个事件（例如，点击鼠标）是不是在 Flash 元素上发生的。点击测试的结果可用来确定是否应将该事件传递到游戏引擎。例如，如果您点击一个 UI 按钮，就可能需要防止游戏进一步处理鼠标点击。此支持是通过 SFManager::DoHitTest 函数提供的。假如该输入事件是在目前显示的任何 Flash 电影上发生的，此函数就返回“真”(True)。

8.1.5 确定当前视窗

确定当前视窗的大小和位置在 Unity 中是一项极大的挑战。Screen.Width/Height 属性提供屏幕宽度/高度，然而，假如重新调整了视窗大小，或者更改了纵横比 (Aspect Ratio)，就不会正确更新这些属性。就我们所知，没有办法在编辑器窗口中获取视窗偏移量。为了克服此局限，我们在本机代码中使用 OpenGL 视窗访问函数，以便在调用显示之前获取视窗大小和位置。如果视窗发生变化，我们就将每个电影所使用的视窗重设为新的视窗尺寸。

可以使用 Movie.bAutoManageViewport 属性来覆盖此行为。如果将此属性设置为“假”(False)，就会始终使用（创建电影时设置的）默认视窗。
请注意，传递到 Scaleform 的所有事件的坐标在内部自动转换到当前视窗，而且不需要任何手动转换。

8.1.6 从 ActionScript 调用 C# 函数

此集成使得声明 C# 函数非常容易，而这些函数可以使用 ExternalInterface 从 ActionScript 进行调用。例如，假如您有一个包含电影剪辑的简单 Flash 文件 Button.swf。您希望每次点击一个按钮时该电影剪辑的位置都根据播放器的位置而发生变化。此逻辑可实现如下：

用 ActionScript 定义一个鼠标点击处理程序：

```
function handleClick(event:MouseEvent):void
{
      if (ExternalInterface.available) {
            ExternalInterface.call("UpdatePosition", player_mc);
      }
};
```

现在在 C# 中，创建一个从 Movie 派生的类，并声明一个将 Value（值）作为一个参数的函数“UpdatePosition”。

```
public class UI_Scene_Movie: Movie
{
      public UpdatePosition(Value movieRef)
      {
            // 使用 Game API 获取播放器位置，使用 GetDisplayInfo 获取与 movieRef 对应的 displayInfo 对象，并在根据播放器位置修改 displayinfo 后使用 SetDisplayInfo 重设 displayinfo。
      }
}
```

在内部，Unity 集成采用 C# 反射找出要在 ExternalInterface 回调期间调用的正确的 C# 函数。为达此目的，您需要在一个 Movie 派生的类中声明一个函数，该类具有与在 ExternalInterface.call 调用中使用的相同的签名（相同名称和参数）。如果找不到匹配的函数，ExternalInterface 调用就会默默地失败。

8.1.7 事件处理

鼠标和键盘事件被用 SFCamera:OnGui 发送到 Scaleform。每个 Movie 可以有自己的视窗，而且在将事件传递到 Scaleform 运行时之前，每个电影在内部执行鼠标位置到电影视窗的转换。通过覆盖 Movie.HandleMouseEvent 可以更改默认转换。您也可以通过覆盖 AcceptMouse/CharEvents 并返回“假”(False) 来预防该电影不响应 Mouse/Key 事件。

8.1.8 在脚本中表示电影

使用一个整数 ID 在脚本中表示电影。此 ID 与 C++ 中的 Movie 指针所表示的整数值相对应。此技术使得通过简单地将 Movie 对象归于一个 Movie 指针来标识用一个 ID 表示的 Movie 对象。

```
SF_EXPORT void SF_HandleMouseEvent(int movieId, float x, float y)
{
      Movie* pmovie = reinterpret_cast<Movie*>(movieId);
      pManager->HandleMouseEvent(pmovie, x,y, icase);
}
```

8.1.9 寿命问题

用 C# 创建的对象的寿命是自动管理的。C# 运行时定期调用垃圾收集器 (Garbage Collector)，以便于销毁那些不具有任何活动引用的对象。Movie 类和 Value 类分别用来表示 Scaleform 电影和 Scaleform

值。因此，C++ 电影和值的寿命与相应的 C# 对象的寿命密切相关。为了确保 Scaleform 电影和值正确销毁，我们覆盖了相应的 C# 类的 Finalize 方法，并手动销毁 C++ 对象。

需要注意的另外一点是，C# 垃圾收集器运行于一个与主游戏线程不同的线程。因此，必须确保调用 C++ Values 和 Movie 方法时线程安全。

8.1.10 Direct Access API

Direct Access API (DAPI) 用来直接访问和修改 Flash 对象及其属性。DAPI 消除了用 ActionScript 创建函数并在每次需要访问 Flash 对象的某个属性时调用它们的必要，因而极大地提高了速度和效率。Scaleform 使用 Value（值）接口表示简单类型（如 Int、Bool）以及复杂类型（如 DisplayObject）。Unity Integration 为整个 DAPI 接口提供一个包装层，这样，用户就可以直接访问用 C# 编写的 Flash 对象。DAPI 使您能够用 C# 编写大多数 UI 逻辑，而不必编写许多 ActionScript 代码。有关更多信息，请参阅 Value.cs 并查阅 StarShip 演示中的 Hud 和“暂停菜单”（Pause Menu）的实现。

8.1.11 跟踪语句

Actionscript 跟踪语句被自动定向到 Unity Console 输出，以便于调试。

8.1.12 部署

在我们的演示中，Flash 素材位于 Assets/StreamingAssets 文件夹。无需修改，此文件夹中的文件会被自动复制到部署平台。

9 iOS 上的特殊考虑因素

在 iOS 平台上进行的开发和部署存在不少差异。本节概述其中许多差异。

9.1.1 的行为

如上所述，pinvoke 用于托管代码与非托管代码之间的通信。在 iOS 与 Windows 平台上实现 pinvoke，二者之间存在明显差异。接下来我们将重点介绍其中几项差异：

1. 在 iOS 上，“代理”(Delegates) 无效：如前所述，在 Windows 上，我们使用代理来从本机代码调用托管代码方法。使用代理是一种方便，因为它们立即执行并可返回值。令人遗憾的是，由于在 iOS Mono 上 AOT（提前）编译产生的局限性，使得代理不工作。为了克服这一局限性，我们将外部接口通知和值参数放到一个共享队列上。此队列按帧轮询，而且就像常规 ExternalInterface 回调一样，处理该队列上存储的任何命令。不过，对 ExternalInterface 回调进行排队意味着不再立即执行这些回调，而是在 Unity 的更新期间进行（有关实现的详细情况，请参阅 SFManager::ProcessCommands()），这使得这些回调不同步。此外，回调无法返回值。我们发现这些局限带来不便，但并不构成多大问题。

2. Marshal.PtrToStructure 方法无效。这意味着，必须手动完成将数据从非托管堆指针封送到类的操作。从用户的观点看，此局限并不十分重要。

10 渲染到纹理

“渲染到纹理”是一种重要的计算机图形技术，可用来创建许多令人关注且赏心悦目的效果。此技术使您能够将您的 Flash 内容拖到一个游戏纹理（而非后台缓冲区）。然后可以将此纹理应用到您的场景中的任意 3D 对象。下面的截屏显示一些 RenderTexture 使用案例。

图 1：绘制有 Flash 电影的 TV 显示器。此 Flash 电影接受键盘输入

图 2：上面有 Flash 电影的另一个 TV 显示器。此对象启用了透明度，因此背景通过未渲染 Flash 内容的区域进行显示。

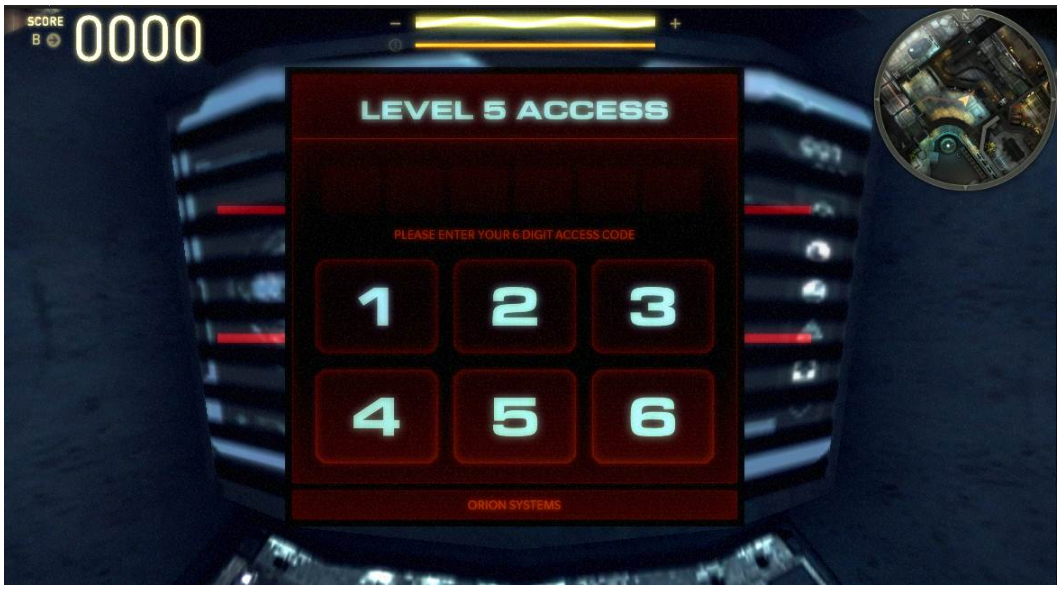


图 3：渲染纹理的另一个示例。该 Flash 电影可以接受用来点击键盘按钮的鼠标输入

10.1 在 Unity 集成中使用渲染纹理

Scaleform-Unity 集成使将您的 Flash 电影设置为渲染到纹理（而不是后台缓冲区）变得非常简单。请按照下面的步骤渲染到纹理。

1. 生成 SFRTT 子类（Plugins\SF\SFRTT.cs 中定义），并添加用来创建 RenderTexture 电影的代码。您可以跟随 HelloWorldDemo 中的 MyRTT.cs 中的代码去看一个示例。如果您的 Flash 电影不发送任何外部接口回调，您可以直接使用 Movie 类，而无需进行子类生成。

2. 如果想要您的渲染纹理电影响应 Flash 事件，如鼠标点击，您就必须在创建 RenderTexture (RTT) 电影的同时覆盖该 Movie 类，并提供子类的名称。这与针对叠加电影的设置相同。

3. 在 Unity 编辑器中，将您在步骤 1 中创建的派生类拖放到您要在其上面绘制纹理的对象。

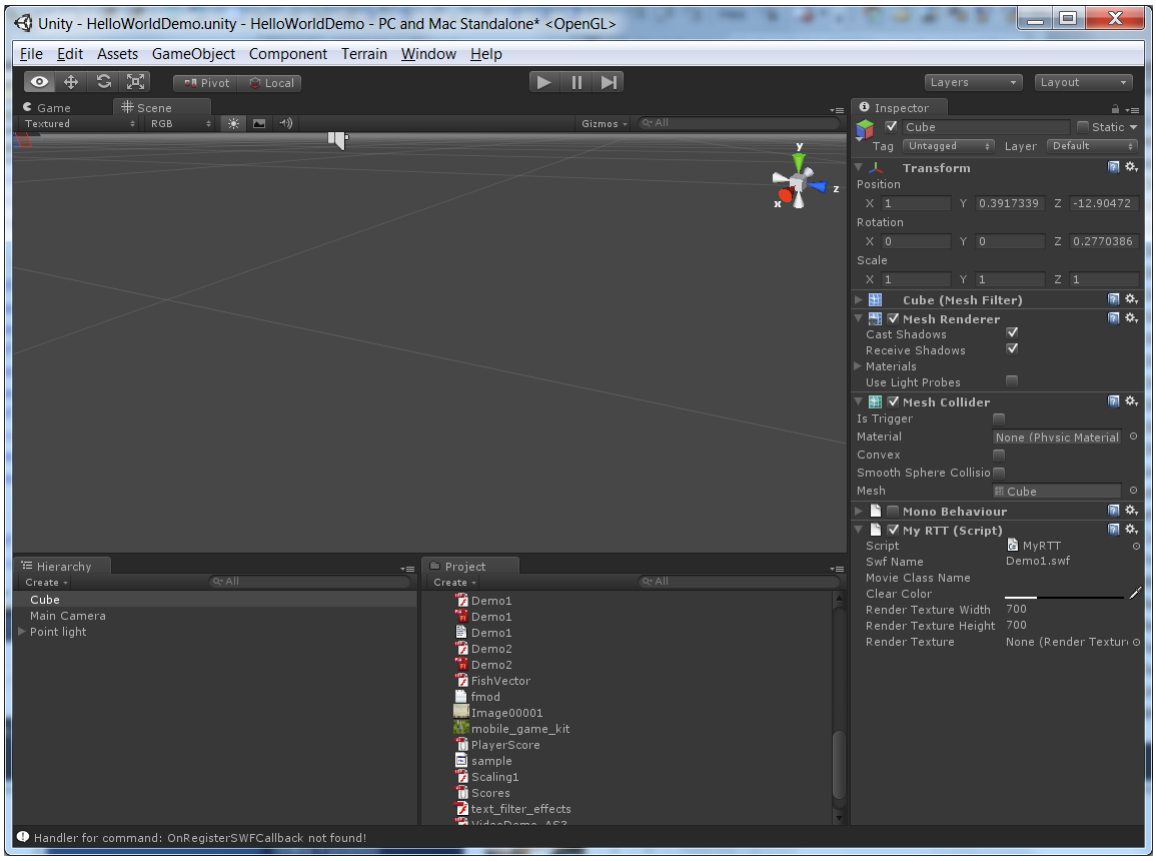


图 4：HelloWorldDemo 中的 RenderTexture 设置

您可以指定实现外部接口回调的 Movie 子类名、Flash 电影的名称（不要忘记 .swf）以及编辑器本身内的渲染纹理的宽度和高度。重要提示：确保宽度和高度相同。否则 RenderToTexture 就不发挥作用。这似乎是 Unity 的一个局限。您也可以为编辑器中的目标纹理指定清晰颜色。

10.2 命中测试

我们实现了将鼠标事件转换为 RTT 电影坐标空间的逻辑（SFMovie.cs 中的 HandleMouseEvents）。此逻辑使用 MeshCollider 组件计算发生鼠标点击的纹理空间中的点。为此，使用 RTT 的 3D 对象必须附加有一个 MeshCollider 组件。

此逻辑旨在证明进行命中测试 (Hit Test) 的一种方法。如有必要，您可以添加自己的自定义命中测试逻辑。从内部来看，Scaleform 仅使用从 C# 传递来的坐标。

10.3 将焦点转移到RTT 电影

SWF 电影需要用来接收用户输入的焦点。电影并不总是能够轻而易举地接收输入，当渲染到一个看不见的纹理时尤其如此。Scaleform 使得控制特定电影是否接收来自用户的输入成为可能。这可以通过切换电影的焦点（SFMovie.cs 中的 SetFocus）来实现。

游戏程序员完全可以确定渲染纹理电影何时适合获得或失去焦点。可以接受命中测试的纹理非常适合接收鼠标输入。换句话说，键盘输入并不具有此种直接包含测试。

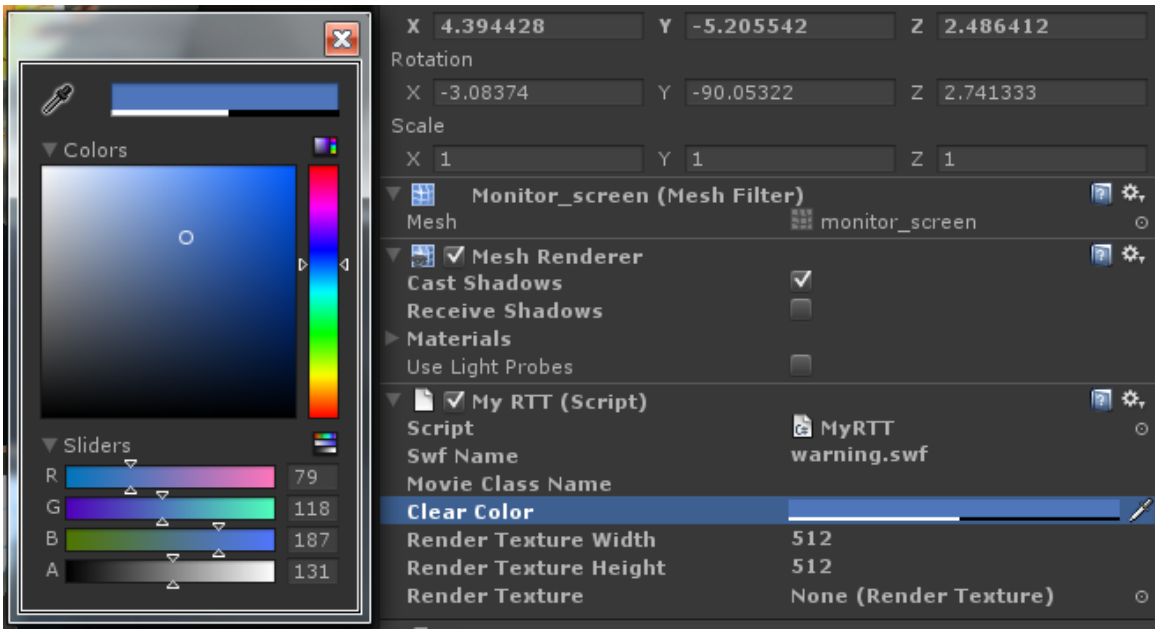
确定是否适合将用户键盘输入发送到渲染纹理的方法有多种。例如，可以通过测试来了解某个对象是否在视图锥截体 (Viewing Frustum) 内。在某些情况下，场景中可以同时存在多个渲染纹理，因而将键盘事件发送到所有可见电影可能并不具有什么意义。因此，在 Starship Down 中，我们选择根据我们的玩家与上述每个渲染纹理之间的距离远近来设定焦点。

10.4 添加阿尔法混合

10.4.1 透明电影

除了 Scaleform 中提供的正常“渲染到纹理”功能之外，还可以毫不费力地拥有阿尔法混合 (Alpha-blended) 的电影。下面的步骤将介绍如何达到此目的：

1. 使用本文前面介绍的指导说明创建一个渲染到纹理对象。
2. 确保该对象的“清晰颜色”(Clear Color) 参数拥有一个不小于 255 的阿尔法值。例如，在下面的图像中，将用一种淡蓝色背景渲染 SWF，该背景具有大约 50% 的透明度。



3. 确保指定给该对象的材料是透明的。例如，可能使用“手机” (Mobile) -> “透明”(Transparent) 类别中的“顶点颜色”(Vertex Color) 材料：

4. Continue enjoying the Scaleform plugin for Unity.

11 附录

11.1 Interop

托管代码和非托管代码通过平台接口服务 (PInvoke) 系统彼此互动。在 Windows 上，系统工作方式如下：

假定您要从 C# 调用函数 C++ `foo(char*)`。首先，在一个 .cpp 文件中实现此函数，并用一个 dll 将其导出。

```
__declspec(dllexport)void foo (char* str)
{
      printf("str\n");
}
```

然后，在非托管代码 (C#) 中，使用 DllImport 属性声明此函数：

```
[DllImport("DllName")]  public static extern void foo(String str);
```

现在，就可以就像一个常规函数一样，从非托管代码调用此函数：

```
foo("Hello World");
```

请注意，pInvoke 封送层负责处理从 C# `String` 到 C++ `char*` 的转换。pInvoke 自动实现简单数据类型的封送，但较复杂的类型可能必须手动封送。例如，DisplayInfo 结构，它封装 Flash 显示对象的显示属性。有关封送和 pinvoke 的详细信息，请参阅 MSDN 说明文档。

现在，我们看看转换情况- 从非托管代码调用托管代码中的函数。要想调用脚本函数以响应 ExternalInterface 回调，就需要执行此操作。在 Windows 上，这是通过代理实现的。

托管代码：

```
// Step 1: Declare a Delegate. A Delegate is similar to a function pointer in C++
public delegate void ReceiveMessageDelegate(String message);

// Exported function to install the delegate
[DllImport("DllName")]  public static extern void
InstallDelegate(ReceiveMessageDelegate del);

// Step 2: The function we intend to call from C++
public void ReceiveMessage(String msg)
{
```

```
      Console.WriteLine("Message Received: " + msg);
}

// Step 3: Create and Install the delegate by passing it to C++
ReceiveMessageDelegate del = new ReceiveMessageDelegate (ReceiveMessage);
InstallDelegate(del);
```

Now consider the C++ (Unmanaged) side:

```
Imported function that receives the delegate:
// Declare a function pointer that has the same signature as the C# delegate
typedef void (ReceiveMessageDelegate*)(char*)
__declspec(dllexport)void InstallDelegate (ReceiveMessageDelegate func)
{
      // Call the delegate just like calling a function from a regular
     // function pointer
      func("Hello World");
}
```

11.2 在 Windows 上构建

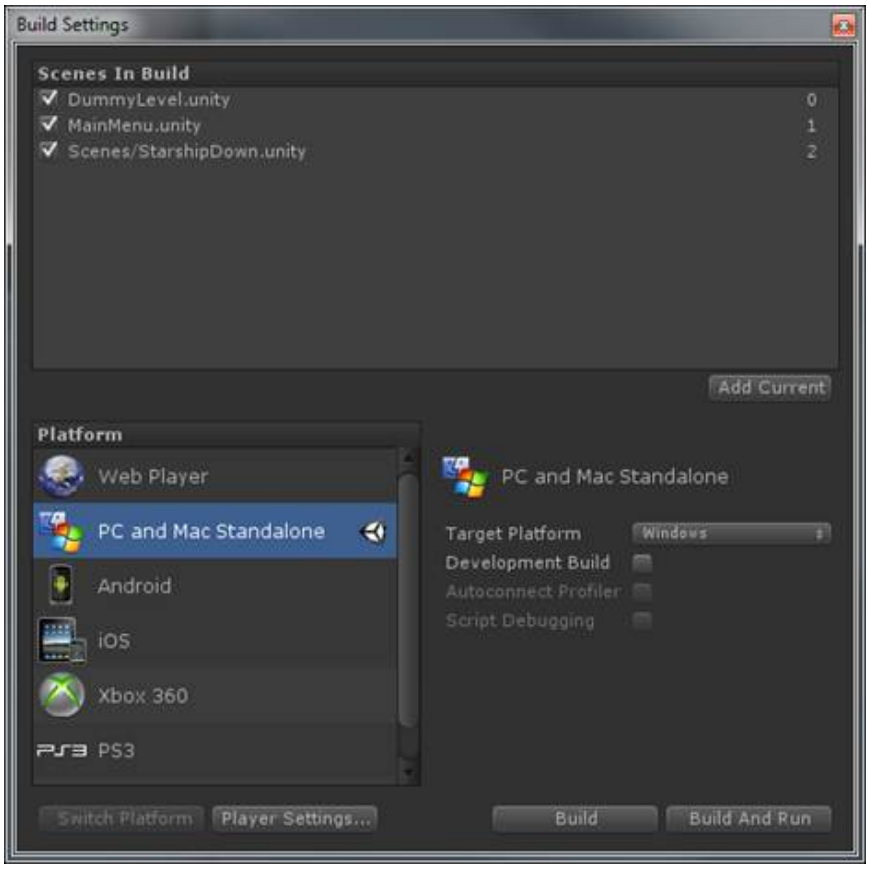
要在 Windows 平台上构建和调试 Starship 演示，请遵循下述指导说明。下一节介绍 iOS 的构建说明。

11.2.1 在不调试的情况下运行演示

1. 从您安装 Unity 的命令提示处（一般为 C:\Program Files (x86)\Unity\Editor）启动 Unity.exe。默认情况下，使用 D3D 渲染器启动 Unity 编辑器。如果想要使用 OpenGL 渲染器，请用命令行选项 “-force-opengl” 启动 Unity.exe。请注意，在 PC 上，HelloworldDemo 和 StarshipDown 中包含的默认 dll 是发布-D3D，因此，要想使用 OpenGL 渲染，就必须从 Unity/Bin 复制适当的 OpenGL dll。

2. 从文件菜单上，浏览到 Integrations\Unity\StarshipDown\Assets 并打开 DummyLevel.Unity。

3. 现在，在“构建设置”(Build Settings)（“文件”(File) ->“构建设置”(Build Settings)) 中，如下图所示，给 DummyLevel、MainMenu 和 StarshipDown 指定关卡级数：

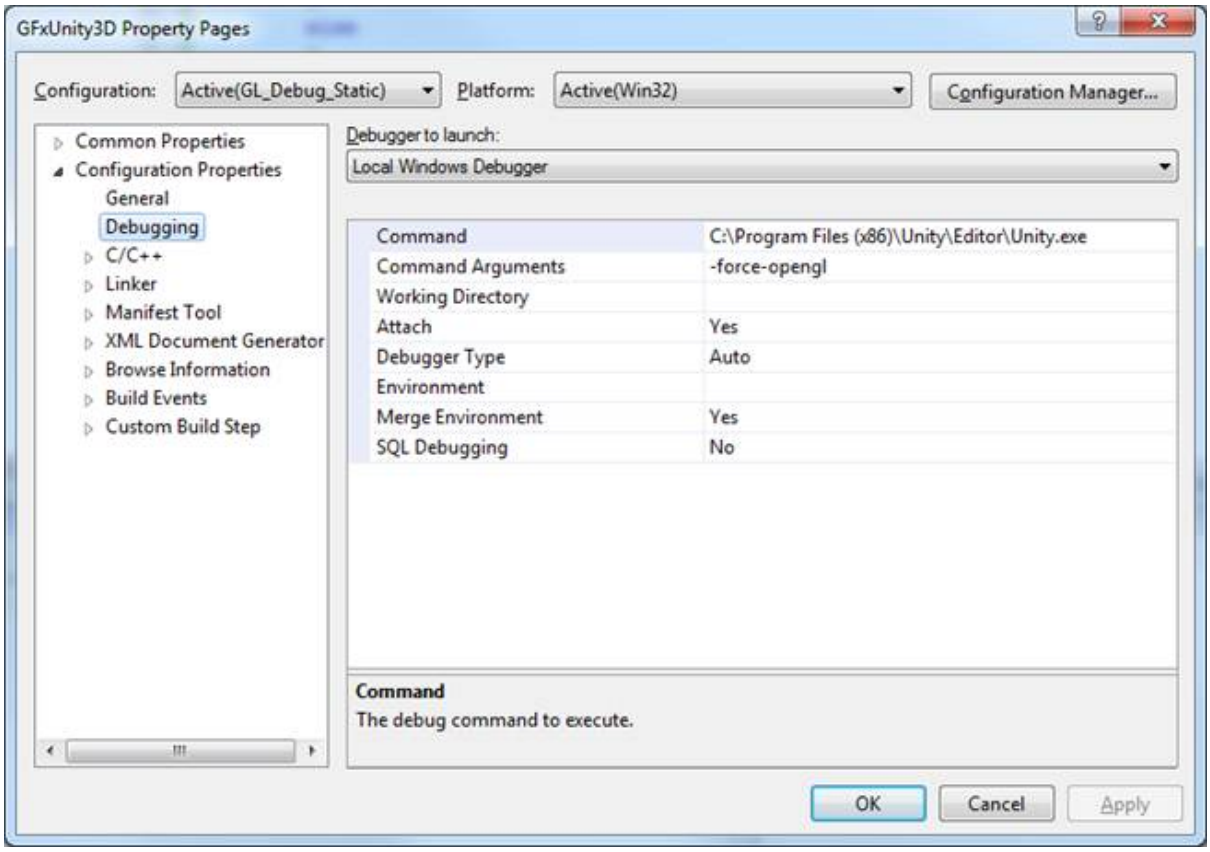


4. 按“播放”(Play)。现在，应该会看到 Scaleform 主菜单。

11.2.2 使用 Visual Studio 进行调试

注意：只有可以访问 Src 的客户才能使用此集成的源代码。假如您想要获得源代码访问权限，请写信给 Scaleform 支持部门。如果您不是源客户，本节与您无关。

从 Integrations\Unity\Projects\Win32\Msvc90 打开 GFxUnity3D，并如下图所示更改“配置属性”（Configuration Properties）：



现在按 Cntrl+F5。这应启动 Unity。请注意，假如您以前没有用 DummyLevel 启动 Unity，就不会是 Unity 启动期间启动的默认关卡，而且必须浏览到手动查找 DummyLevel 的文件夹（如上所述）。

要连接到调试程序，请按 F5，现在您应该能够在集成代码中设置断点了。

11.3在 iOS 上构建

在 iOS 上构建 Unity 应用程序的过程与在 Windows 上有很大差异。最大的差别是 Scaleform 插件实际链接到您的应用程序，而不是用作一个 dll。因此，必须用 DllIlmport("__Internal") 代替 C# 文件中的 DlllImport("libgfxunity3d") 语句。为使此操作更加容易和透明，我们将对使用导入的函数的类的实现拆分为两个文件。例如，把对 Movie（电影）的实现拆分成 SFMovie.cs 和 SFMovie_Imports.cs。Imports 文件包含 ifdef's，这样，就会自动使用版本正确的导入函数。

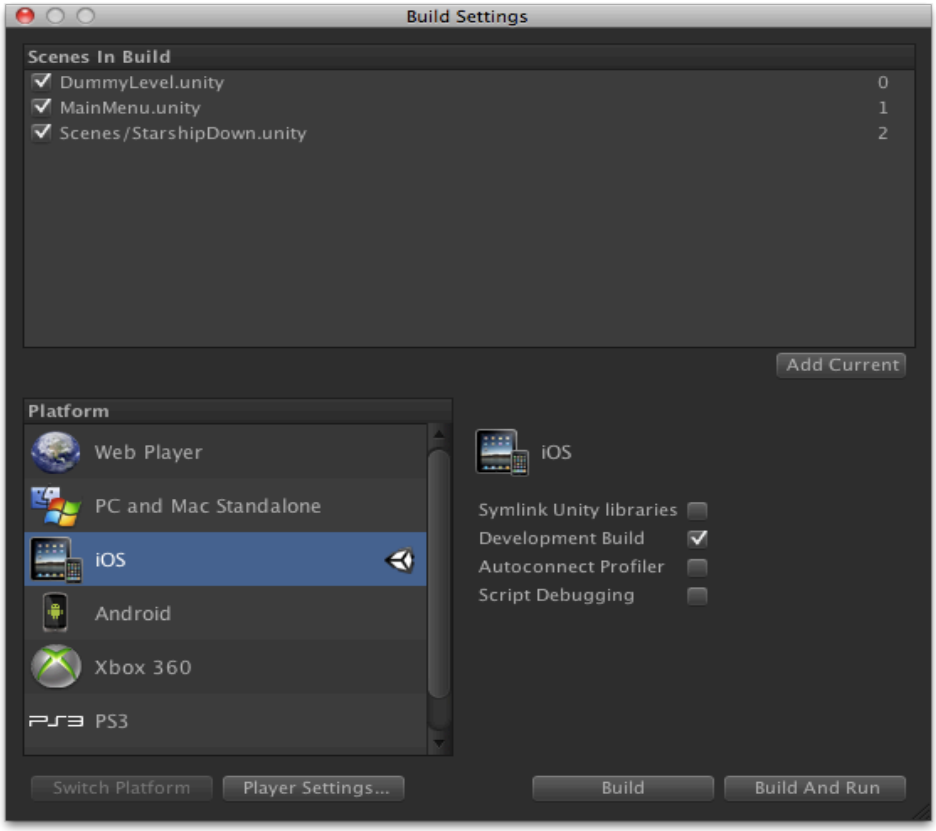
Unity 生成此演示的 Xcode 项目，然后通过添加集成代码和 Scaleform 库在 Xcode 内进行修改，以支持 Scaleform。下面的路径下的集成程序包中包含有用于 Starship Down 演示的 iOS Xcode 项目的一个预构建副本：

- StarshipDown/iOS Project/

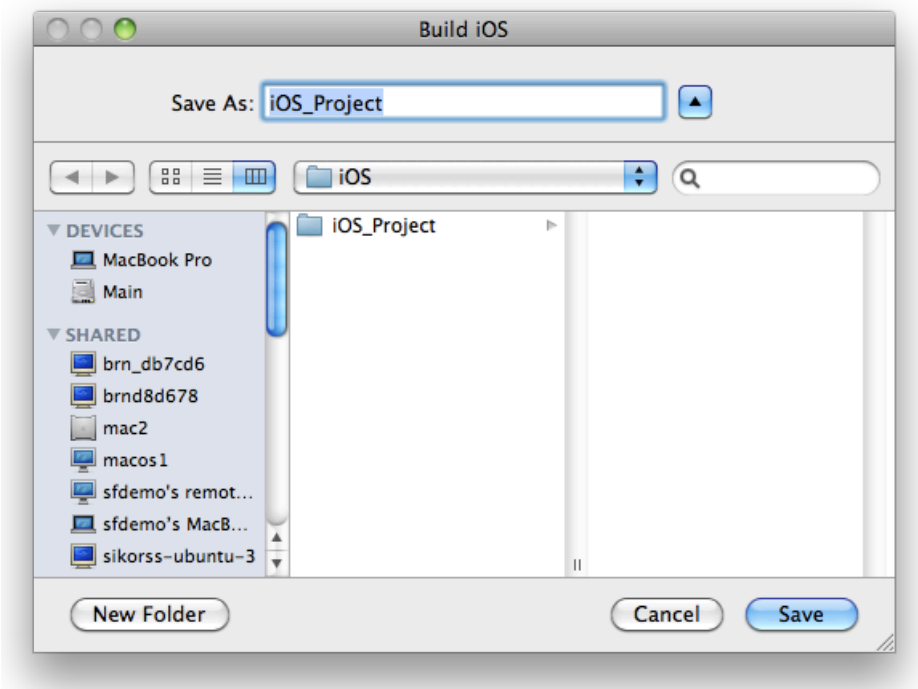
此 StarshipDown 项目为客户提供一个参考，用来在 Scaleform 支持的情况下配置自己的 Unity Xcode 项目。为达此目的，将集成代码、Scaleform 库、素材等添加到 Xcode 项目，并更新了该项目的“项目设置”(Project Settings)。这些都有助于有效地将此集成编译到您的游戏应用程序之中，而且是 iOS 所必需的过程。

无论任何 Unity 内容何时发生变化（“场景”(Scene)、“网格”(Mesh)、“材料”(Material)、“Unity 脚本”(Unity Script) 等），在 iOS 设备上查看 Xcode 项目之前，都需要对其进行更新。您可以使用下面的步骤更新该项目：

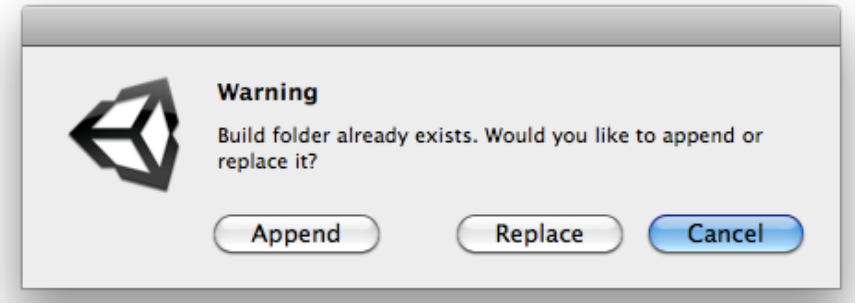
1. 选择“文件”(File) -> “构建设置”(Build Settings)



2. 确保“构建中的场景”(Scenes In Build) 中列出的“场景”(Scenes) 正确，且排列顺序正确。
3. 在“平台”(Platform) 下，选择 iOS。
4. 单击“构建”(Build) 按钮。
5. 提示保存文件的位置时，选择适当的位置（一般情况下，该位置是一个现有 Xcode 项目的位置）。



6. 提示是要“附加”(Append)、“替换”(Replace) 还是“取消”(Cancel) 时，选择“附加”(Append)。请注意，选择“替换”(Replace) 将会导致重新构建整个 Xcode 项目，删除可以配置的任何 Xcode “项目设置”(Project Settings)。



7. 打开 Xcode 项目。
8. 从 Xcode 构建并运行/调试该项目。

假如这是您首次在 iOS 上运行“项目”(Project)，请注意，您必须在 iOS 的“Unity 项目设置”(Unity Project Settings) 内安装一个有效的“供给配置文件”(Provisioning Profile) 和“捆绑标识符”(Bundle Identifier)。有关正确配置供给配置文件和捆绑标识符的更多信息，请访问 Unity 的网站和 Apple 开发者支持网站。

要配置自己的 Xcode 项目以支持 Scaleform 4.2，请确保使用下列设置：

Unity 播放器设置 ：

目标平台：armv7

SDK 版本：iOS 5.x or 6.x

目标 iOS 版本： 5.x or 6.x

这很重要，否则您在 iOS 上部署应用程序时就可能会收到下面的错误：

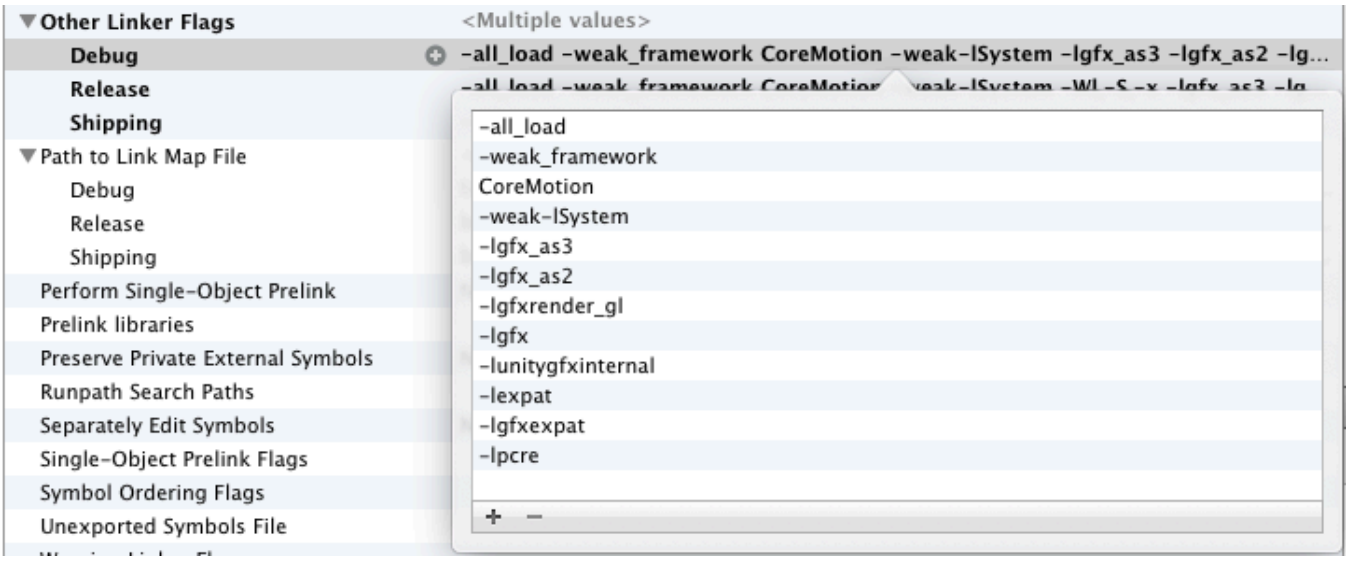
"Uncaught Exception:
*** -[PBXDebugScriptCommand debugSessionDidStart:]: unrecognized selector sent to instance 0x49726c0"
（ “未捕获到异常：
*** -[PBXDebugScriptCommand debugSessionDidStart:]: 无法识别向实例 0x49726c0 发送的选择器”）

要添加到 Xcode 项目的文件：

- Scaleform 插件代码：
    - {SDK}/Integrations/Unity/Src/SFExports.cpp

- Scaleform 库（针对各种配置，将这些添加到您的 Target 的构造设置中 – 我们将在下面的示例中使用 Debug）

    - {SDK}/Lib/iPhone-armv7/Debug_NoRTTI/libgfx.a
    - {SDK}/Lib/iPhone-armv7/ Debug_NoRTTI/libgfx_as3.a
    - {SDK}/Lib/iPhone-armv7/Debug_NoRTTI/libgfx_as2.a
    - {SDK}/Lib/iPhone-armv7/Debug_NoRTTI/libgfxexpat.a
    - {SDK}/Lib/iPhone-armv7/Debug_NoRTTI/libgfxrenderer_gl.a
    - {SDK}/Lib/iPhone-armv7/Debug_NoRTTI/libgfxexpat.a
    - {SDK}/Lib/iPhone-armv7/Debug_NoRTTI/libexpat.a
    - {SDK}/Lib/iPhone-armv7/Debug_NoRTTI/libpcre.a
    - {SDK}/Integrations/Unity/Lib/iOS/Debug-iphoneos/libunitygfxinternal.a



-所有必需的 .SWF 文件（例如，UI_HUD.swf、UI_LoadingScreen.swf 等）应包含在您的项目的 StreamingAssets 文件夹中，例如：
{SDK}/Integrations/Unity/StarshipDown/Assets/StreamingAssets/

Xcode 项目设置：

构建设置：

目标平台：armv7

构造选项：

用于 C/C++/Objective-C 的编译器：LLVM GCC 4.2

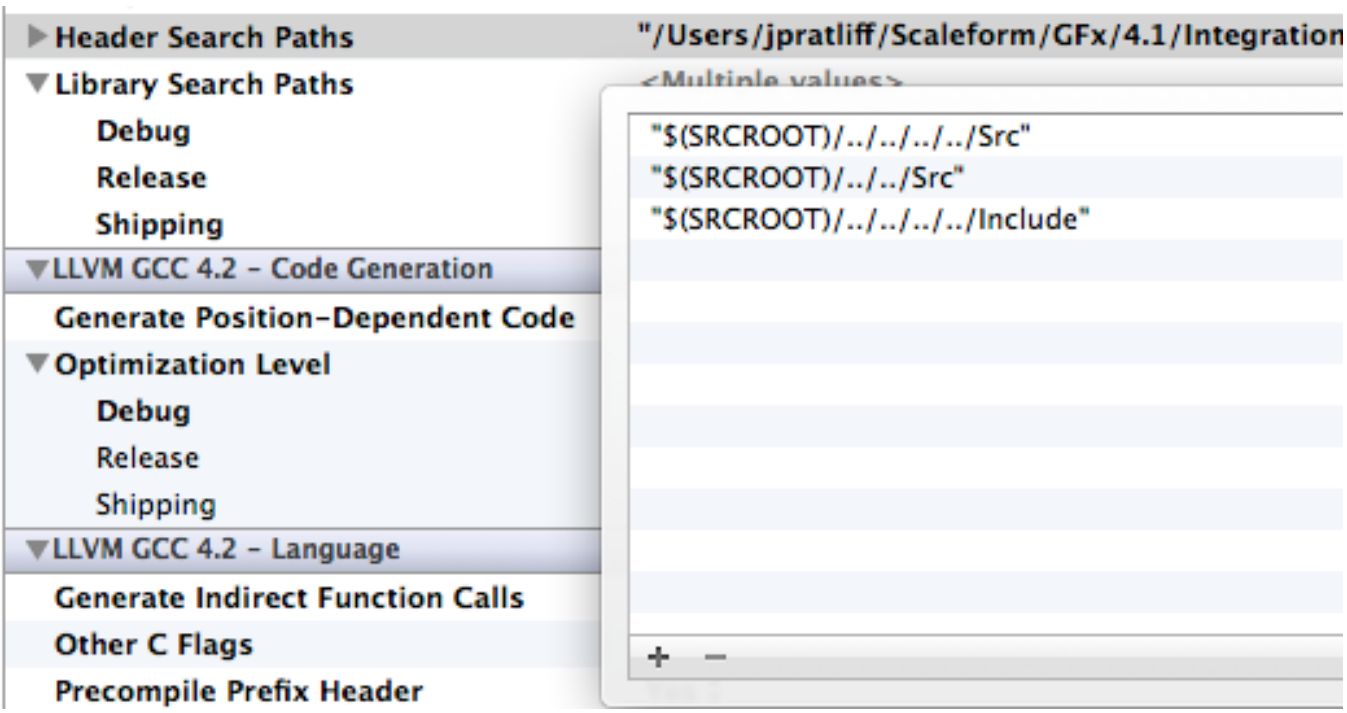
代码生成：

缩略图编译：假（False)

搜索路径：

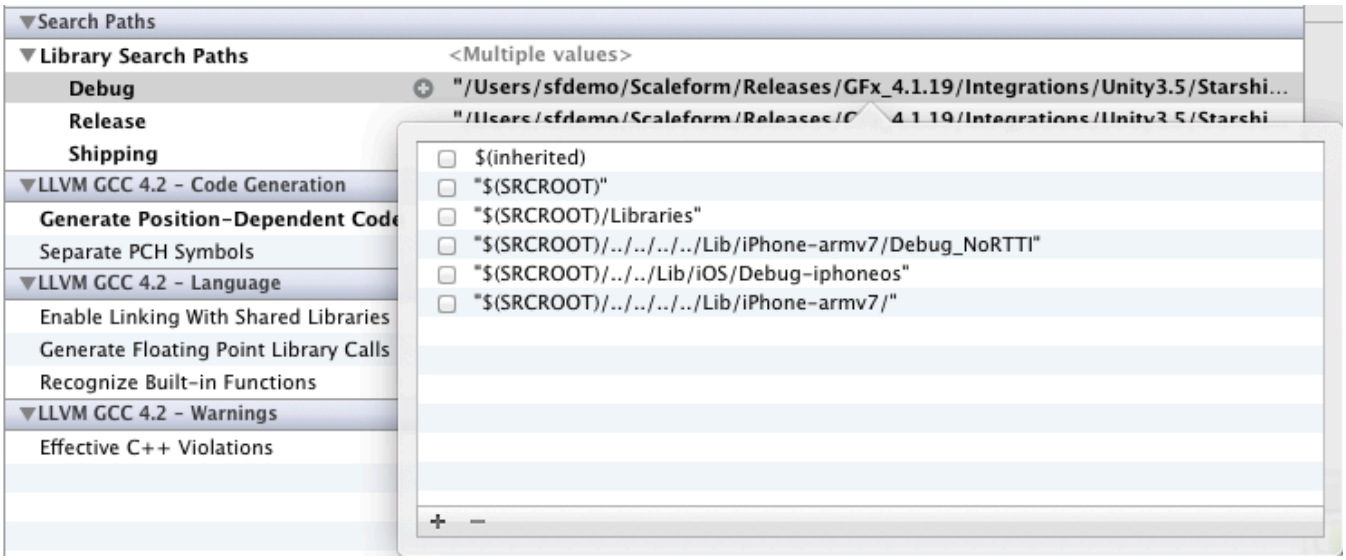
标题搜索路径：

- {SDK}/Integrations/Unity/Src/
- {SDK}/Src
- {SDK}/Include



库搜索路径（我们将使用 Debug Lib 路径作为示例：

- {SDK}/Lib/iPhone-armv7/
- {SDK}/Lib/iPhone-armv7/Debug_NoRTTI
- {SDK}/Integrations/Unity/Lib/iOS/Debug-iphoneos/



GCC 4.2 – 语言

启用 C++ 例外：No

启用 C++ 运行时类型：No

其它 C 标志/其它 C++ 标志：-DSF_BUILD_DEBUG（用于调试用内部版本）

LLVM GCC 4.2 – Preprocessing

默认情况下，Unity 不在针对 iOS 的 OpenGLES2 实现中提供一个模板附件 (Stencil Attachment)。这可防止 Scaleform 绘制掩码。要添加掩码支持，您需要在 iPhone_GlesSupport.cpp ({SDK}/Integrations/Unity/StarshipDown/iOS_Project/Classes/) 的第 145 行进行如下代码更改。请注意，更改以黄色突出显示：

```
if(surface->depthFormat)
{
      GLES_CHK( glGenRenderbuffersOES(1, &surface->depthbuffer) );
      GLES_CHK( glBindRenderbufferOES(GL_RENDERBUFFER_OES, surface->depthbuffer) );
      GLES_CHK( glRenderbufferStorageOES(GL_RENDERBUFFER_OES, 0x88F0, surface->w,
surface->h) );

      UNITY_DBG_LOG ("glFramebufferRenderbufferOES(GL_FRAMEBUFFER_OES,
GL_DEPTH_ATTACHMENT_OES, GL_RENDERBUFFER_OES, %d) :: AppCtrl\n", surface-
>depthbuffer);
      GLES_CHK( glFramebufferRenderbufferOES(GL_FRAMEBUFFER_OES,
GL_DEPTH_ATTACHMENT_OES, GL_RENDERBUFFER_OES, surface->depthbuffer) );
      GLES_CHK( glFramebufferRenderbufferOES(GL_FRAMEBUFFER_OES,
GL_STENCIL_ATTACHMENT_OES, GL_RENDERBUFFER_OES, surface->depthbuffer) );
}
```

11.4 在 Android 上构建

在 Android 上运行您的支持 Scaleform 的 Unity 应用程序与在 Windows 相似。不需要编译任何 No C++ 代码。如前所述，Scaleform 运行时包含在位于 Unity/Bin/Android 的 libgfxunity3d.so 共享对象文件中。为了方便起见，这些文件也复制到 HelloWorldDemo 和 StarshipDown 演示的 Assets/Plugins/Android 目录。

11.4.1 运行 HelloWorldDemo

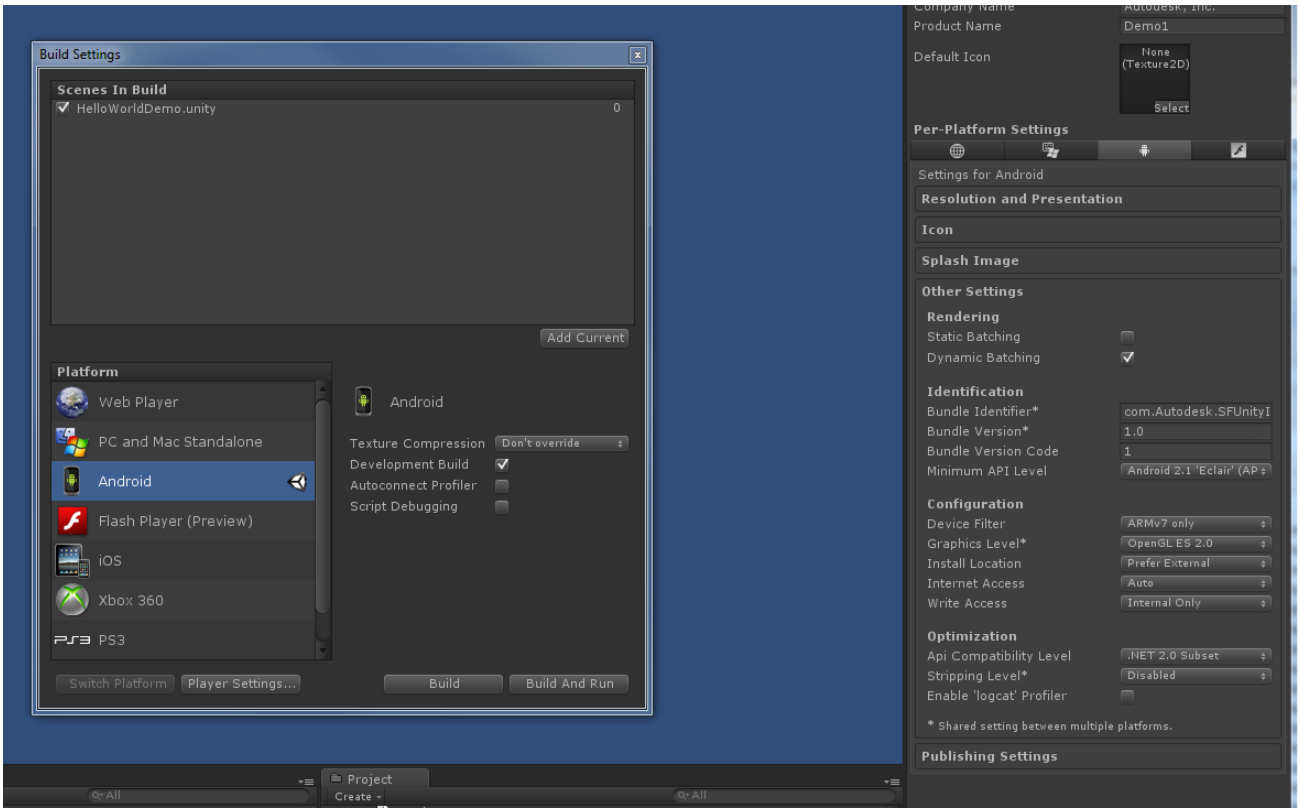
打开 Unity 编辑器，并浏览到 HelloWorldDemo/Assets/。打开 HelloWorldDemo.unity 并转到构建设置，并选择 Android 作为目标平台。 点击“构建和运行”(Build and Run)。如果一切设置正确（请查阅下面有关播放器设置的说明），Unity 应构建 HelloWorldDemo.apk 并在设备上启动该程序。

11.4.2 创建自己的应用程序

创建 Android 应用程序的关键步骤是：

1. 将核心集成脚本文件复制到 Assets\Plugins\SF 中
2. 创建一个覆盖 SFCamera 的脚本，并将其附加到您的场景中的一个相机对象
3. 将 libgfxunity3d.so 复制到 Assets\Plugins\Android 文件夹
4. 确保您的 Flash 素材位于 Assets\StreamingAssets

现在，在 Unity 编辑器中，切换到 Unity 平台，并确保按下图所示设置播放器设置。



重要设置：

设备过滤器（Device Filter): ARMv7

图形等级 (Graphics Level): OpenGL ES 2.0
最低 API 等级（Minimum API Level): API level 7

11.4.3 使用 cygwin 在 Android 设备上启动 .apk

第 1 步：浏览到 .apk 所在的目录。
第 2 步：在 Cygwin 命令窗口上，键入：`adb install HelloWorldDemo.apk`
第 3 步：要看到调试日志，请使用：**`$ adb logcat -s Unity ActivityManager PackageManager dalvikvm DEBUG`**