Autodesk®
Scaleform®

Unity Scaleform 集成概述

本文介紹 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

本文介紹 Unity-Scaleform 集成的主要特性。我們假設使用者初步瞭解採用 Adobe Flash(電影剪輯、事件等)進行的 UI 設計,並且對使用 Scaleform GFx(Advance/Display/ExternalInterface 等)有所熟悉。如果您不熟悉 Flash 和/或 Scaleform,請參閱我們的評估套件中包含的《Scaleform 4.2 入門》(Getting Started with Scaleform 4.2)。

本文其餘部分的內容如下：

- 「安裝」介紹安裝套裝程式後需要執行什麼操作。
- 「發行版本」(Distribution) 介紹此套裝程式中包含什麼內容。
- 「演示」介紹隨此套裝程式分發的 HelloworldDemo 和 StarshipDown 演示。請注意,附錄中提供有在 PC/iOS/Android 上構建這些演示的詳細說明。
- “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」詳細介紹在您自己的應用程式中使用此集成時所需執行的步驟。
- 「使用此集成」詳細介紹在您自己的應用程式中使用此集成時所需執行的步驟。
- 最後一節概述與此產品相關的 Windows、iOS 和 Android 之間的區別。這一節對 Android/iOS 開發者來說尤其重要,因為本文絕大部分內容是從 Windows 使用者的角度編寫的。

附錄中提供了有關 Windows、iOS 和 Android 平臺的詳細構建說明。

2 安装

如果在安装過程中選擇預設值,此集成就應安裝在 Documents\Autodesk\Scaleform\SF4.2_Unity 中。本文中的任何相關路徑均引用此根目錄。

安裝後的第一步是導入 Integrations\Unity\HelloworldDemo 和 Integrations\Unity\StarshipDown 資料夾中封裝的素材 (Asset)。為此,您可以按兩下 HelloworldDemo.unitypackage/StarshipDown.unitypackage 檔,也可以創建新的 Unity 專案並導入套裝程式。對於 HelloworldDemo,我們建議將素材導入 HelloworldDemo 資料夾,這樣,導入完成之後,您就會擁有下面的目錄結構：

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

這樣一來,您的資料夾結構就與本文其餘部分顯示的資料夾結構相一致。

3 发行版

此發行版本包括下列內容：

Doc：包含一個全面的說明文檔,其內容涵蓋我們 SDK 的各個方面。如果您不熟悉 Scaleform,就最好從位於 Doc/GFx 的「入門」(Getting Started) 指南開始。您還應熟悉我們的分析工具 – AMP(記憶體及性能分析器)。

Bin：

- Win32 或 MacOS:用於 PC/Mac 的浮水印版播放機可執行程式 (GFxMediaPlayer.exe)。您可以使用此可執行程式在開發平臺 (PC/Mac) 上運行 Flash 檔,然後再將這些檔與 Unity 一起使用。一般情況下,只能將 Flash 檔拖放到播放機視窗上。請注意,此發行版本不包含可在行動裝置上啟動的播放機可執行程式。
- Data/AS3：可用於測試用途的一些示例 SWF/FLA 檔。
- AmpClient.exe 是我們的分析器工具,可用來獲取有關 Unity 編輯器中或行動裝置上運行的 Scaleform 內容的詳細的幀級渲染和記憶體統計資料。Exporter.exe 是匯出應用程式,用來從 swf 檔創建壓縮的 gfx 檔。有關這些應用程式的更多資訊,請參閱說明文檔。

Lib：包含在 iOS 上構建您的應用程式所必需的浮水印 Scaleform 庫。PC/Android 上不需要

資源：包含我們的 CLIK(通用輕便介面工具箱)庫,該庫包含常用 UI 元件(如按鈕、滾動清單等)的預裝實現。有關 CLIK 入門的更多資訊,請參閱我們的說明文檔。此發行版本包含的演示廣泛使用 CLIK,我們鼓勵您熟練使用 CLIK,因為這會使 UI 開發更加方便快捷。

Integrations/Unity/Doc：此文檔的主機資料夾

Integrations/Unity/Bin：包含 Scaleform 二進位檔案的 Debug/Release/Shipping 版本。這些二進位檔案在 PC 上是動態連結程式庫 (dll),而在 Android 上是共用物件 (.so)。對於 iOS,請參閱下面的「Integrations/Unity/Lib」。將這些二進位檔案的發佈版本也複製到特定專案的資料夾(例如,HelloworldDemo/Assets/Plugins/PC 和 HelloworldDemo/Assets/Plugins/Android)之中。在編輯器中按「播放」(Play) 或以獨立模式運行您的應用程式時,Unity 就會自動載入動態庫。

Integrations/Unity/Lib：包含 iOS 上的 GFxUnity3DInternal 的 Debug/Release/Shipping 版本。此 lib 是用於 Unity 與 Scaleform 之間的介面的一個包裝器 (Wrapper),用 XCode 構建您的 iOS 應用程式時,就需要將此 lib 與核心 Scaleform lib 連結在一起。

由於 PC/Android 使用動態連結程式庫,此資料夾在那些平臺上是空的。

Integrations/Unity/Src：包含 SFExports 以及在 iOS 上構建您的應用程式所需的一些其它檔。PC/Android 上不需要。

Integrations/Unity/

HelloWorldDemo

StarshipDown: 我們的演示專案,支援所有平臺。

現在,我們就介紹一下在演示專案(以 HelloworldDemo 為例)中分發 Scaleform 素材的情況。請注意,您必須首先導入隨此發行版本一起提供的 HellworldDemo/StarshipDown.unitypackage 檔。

每個專案使用的 Flash 素材(SWF/FLA 檔)均位於 HelloWorldDemo\Assets\StreamingAssets。部署時,Unity 自動將此資料夾中的檔案複製到目標平臺上。

此集成中的 C# 腳本位於 HelloworldDemo\Assets\Plugins\SF 和 HelloworldDemo\Assets\Scripts\Scaleform 目錄。第一個資料夾包含實現該外掛程式的核心功能的腳本。這些腳本在 StarshipDown 與 HelloworldDemo 之間通用。如果創建一個使用 Scaleform 的新 Unity 應用程式,您就只能將這些腳本複製到一個相應的位置。第二個資料夾包含針對應用程式的代碼的腳本。

Scaleform 二進位檔案在 Android 和 PC 上分別位於 HelloworldDemo\Assets\Plugins\Android 和 HelloworldDemo\Assets\Plugins\PC。如前所述,此發行版本包含位於 Unity\bin 的這些二進位檔案的 Debug/Release/Shipping 版本。為了方便起見,也將這些二進位檔案的發佈版本複製到每個演示的 Plugins 資料夾中。

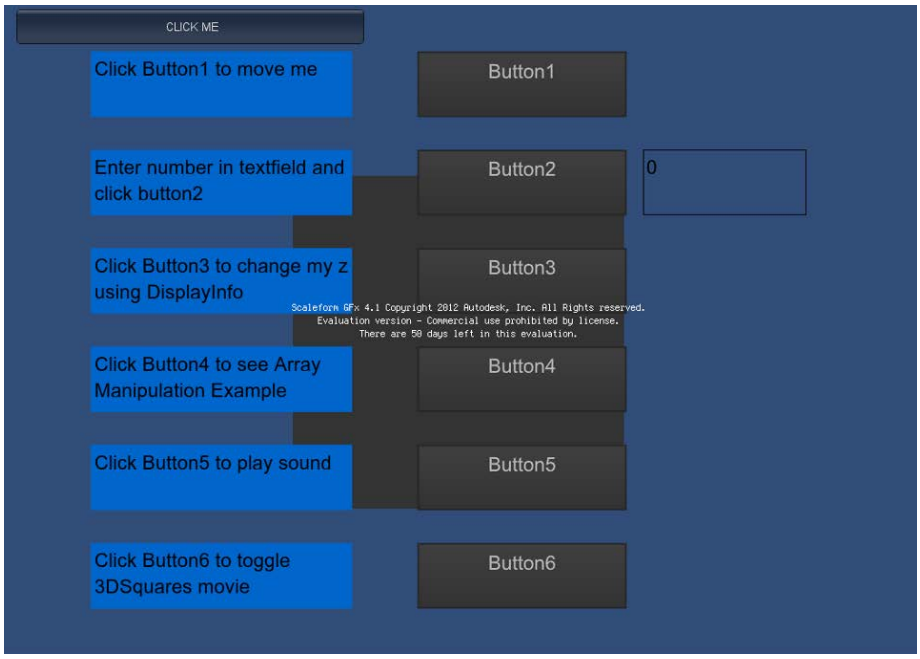
在 iOS 上, 您將使用位於每個演示的 iOS_Project 資料夾的 XCode 專案來構建您的應用程式。此專案連結 Scaleform lib 以及構建您的 iOS 應用程式所需的所有其它 Unity 和系統 lib。

儘管 Unity 支援多種指令碼語言,迄今為止我們還是用 C# 實現我們的集成。我們認為 C# 對於 Unity 來說是使用最廣泛的指令碼語言。如果想在您的腳本編寫中使用 JS/Boo,把 C# 邏輯轉換為其它語言應該是非常簡單的。

4 演示

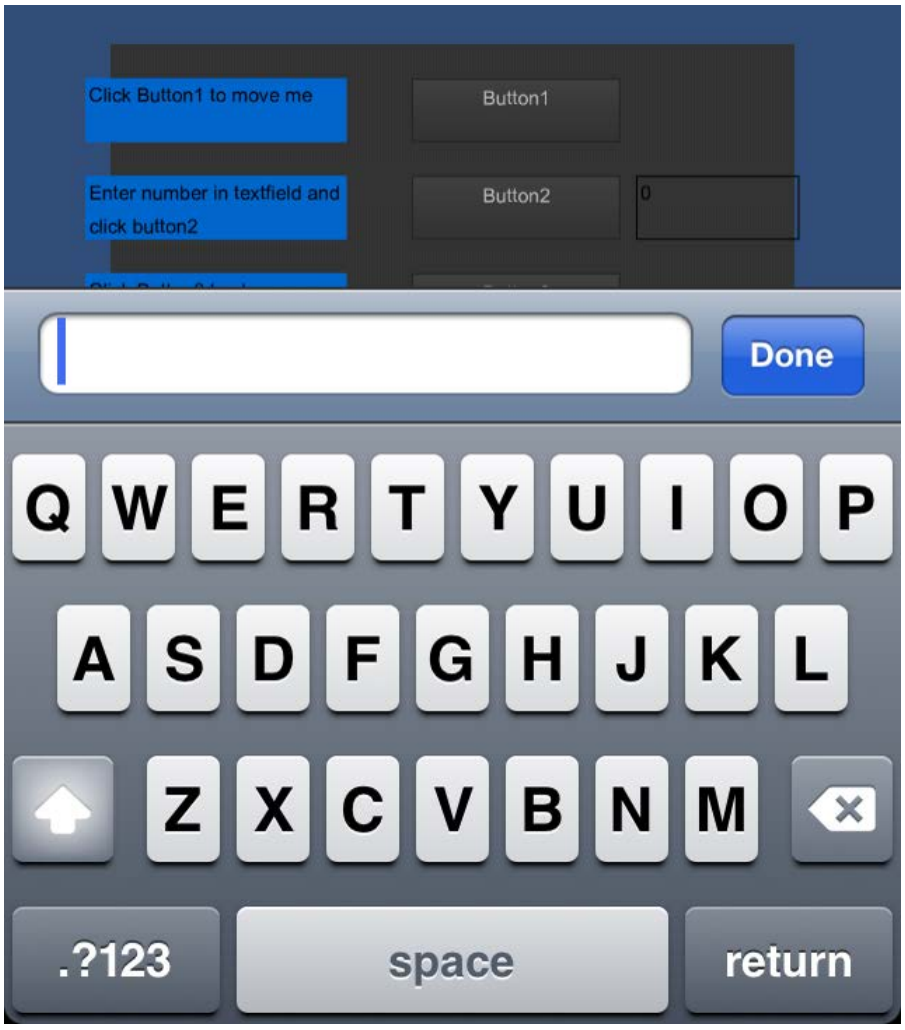
4.1 HelloWorldDemo

此簡單但有用的演示顯示使用 Scaleform 的許多重要方面。此演示的代碼包含在 Demo1.fla、MyCamera.cs 和 UI_Scene_Demo1.cs 中。



1. 使用 Direct Access API 修改 Flash 物件的屬性:按一下按鈕 1 可向左/右移動矩形,而按一下按鈕 3 則可更改矩形的 z 值。這還可以演示 3Di,即將一個 z 值指定給 Flash 電影剪輯的能力。

2. 在不同平臺上輸入文本:按一下文字欄位,並鍵入一個數位。現在,當您按一下按鈕 2 時,後臺中的立方體將以一個角速度旋轉,該角速度與文字欄位中鍵入的數位相一致。在移動平臺上,當該文字欄位收到焦點時,會彈出一個虛擬鍵盤,您可以鍵入數位。



3. 在 Actionscript 與 C# 之間來回傳遞複雜資料:當您點擊按鈕 4 時,就會將一組結構(包含名稱和核心)傳遞到 C#,在那裡可以對其進行修改。這將表明實現記分牌/排行榜所需的管道。

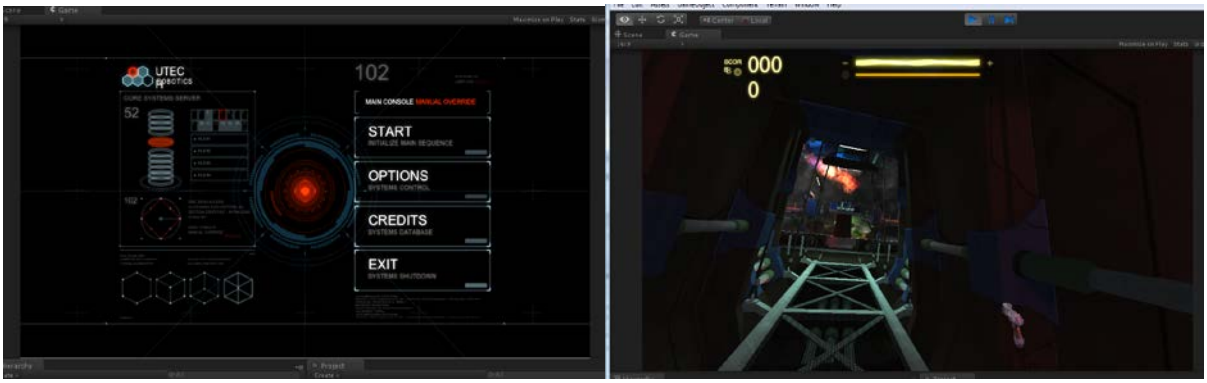
4. 播放聲音:點擊按鈕 5,就會聽到 electric.wav,這是嵌入 demo1.swf 的一個波形檔 (Wave File)。此演示使用 fmod 播放聲音,並且目前僅支援 PC。

5. 通過 Unity 系統聲音播放聲音:當您按一下按鈕 7 時,就會聽到「AutoTurretFire.wav」 – 一個外部波形檔。此演示使用 Unity 的本機系統聲音播放支援。

6. 用 C# 創建和銷毀電影:點擊按鈕 6 可切換 3Dsquares 電影。

4.2 StarshipDown Demo

StarshipDown 演示用圖說明製作一些常用 UI 螢幕(如主功能表、HUD 和暫停功能表),並顯示 HUD 上的簡單遊戲的結果。StarShipDemo 從一個主功能表 (Main Menu) 螢幕開始。一按 START(開始),就會非同步載入遊戲關卡。載入遊戲關卡時,會顯示一個載入屏。當 StarshipDown 關卡完成載入時,會顯示一個得分和生命值/彈藥 (Health/Ammo) 欄,作為 HUD 的組成部分。得分/生命值資料在您演練該關卡和射擊惡魔時會發生變化。按 Escape,就會顯示一個 PAUSE(暫停)功能表。



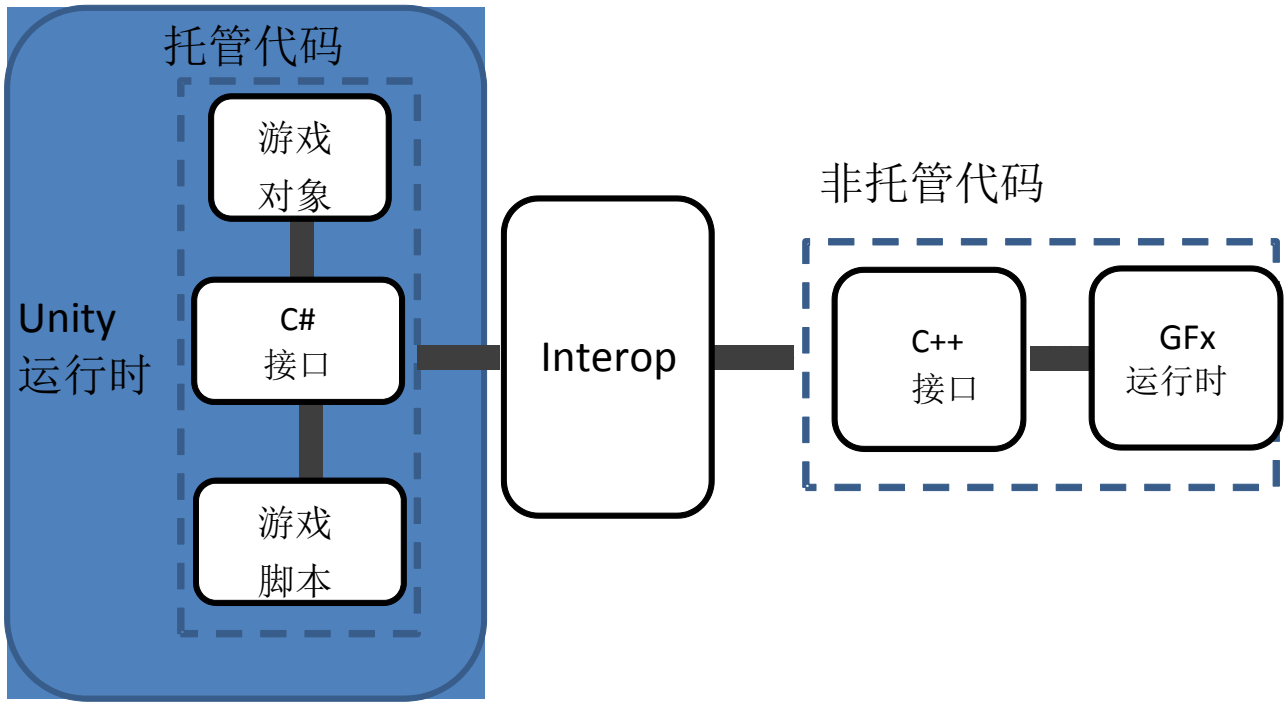
5 当前实现中的局限性

目前,僅支援渲染到主框架緩衝區。尚未實現渲染到紋理。也不支援多層渲染。在此集成的未來版本中將會實現這些功能。

6 架构

在 Windows 上,此集成包含一個 dll,它包裝 Scaleform 運行時,並匯出可從腳本層調用的一系列函數。在 iOS 上,Scaleform 庫連結到此應用程式中。Android 與 Windows 相似,它將 Scaleform 運行時包含在一個共用物件檔中。

儘管 Unity 支援許多指令碼語言(C#、JS、 Boo),我們還是僅提供腳本層的 C# 實現。下面的圖表顯示集成結構。



此集成有兩個主要元件。

- C# 層(託管代碼):包含 SFManager 類的 C# 實現。此層負責具現化 C++ 層和 Scaleform 運行時、調用 Advance/Display、將滑鼠/按鍵事件傳遞到 Scaleform、從 Scaleform 接收回調,並與遊戲中的各種腳本通信。

- C++ 層(非託管代碼):此層充當 Scaleform API 的包裝器,並執行與 C# 層通信所需的一些封送(Marshaling)。

附錄中的「Interop」一節將會更加詳細地介紹託管代碼與非託管代碼之間的通信。

7 使用集成

本節列舉在 Unity 中使用 Scaleform 時需要進行的關鍵步驟。

第一步是創建一個遊戲物件,並將一個腳本附加到它上面,該遊戲物件初始化 Scaleform 運行時並具現化 Scaleform 管理器 (Scaleform Manager)。「Main Camera」 遊戲物件在 StarshipDown 演示中執行此角色。將「ScaleformCamera」 腳本附加到此物件。Scaleform Camera 與其基類 SFCamera 一起執行初始化 Scaleform 並與 Unity 事件系統掛鉤所需的所有步驟。

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。

第二步是通過具現化 Movie(電影)派生的類來創建 Scaleform 電影。Movie 已經包含執行與電影相關的各種任務(如創建/銷毀電影、通過與 Scaleform 外掛程式互動來調用「推進」(Advance)/「顯示」(display))所必需的核心功能。

第三步是在您的 Movie 派生的類中添加自訂事件處理常式。通過在 C# 中使用反射,此集成使得添加事件處理常式非常容易。您只需添加具有與相應的 externalInterface 回檔相同的名稱並接受相同的參數的函數。StarshipDown 演示廣泛利用此模式。例如,查看 UI_Scene_HUD.cs 檔。

在下一節中,我們重點介紹在使用此集成時應瞭解的許多關鍵考慮因素。

8 关键考虑因素

8.1 多线程架构

Unity 引入一個多執行緒渲染器,因而渲染和遊戲更新在不同執行緒中發生。此架構與 Scaleform 4.1 巧妙地配合在一起,而 Scaleform 4.1 對於「推進」和「顯示」也使用不同的執行緒。在多執行緒架構中,必須在渲染執行緒中創建所有與渲染相關的資源(如 HAL)。由於通常通過從遊戲腳本(在主遊戲執行緒上運行)調用匯出的函數來驅動外掛程式,因而不能從腳本發出任何與渲染相關的調用。為了實現讓像我們的執行渲染的外掛程式這樣的低級外掛程式正常工作,Unity 引入某種新的 API,這種 API 從渲染執行緒調用一個匯出的函數,該函數具有一個預定義的名稱和簽名。我們使用此 API 同時支援 D3Dx 以及多執行緒渲染。

有關此話題的更多資訊,請參閱 Unity 說明文檔：
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）

另請參閱：
`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 方法可用來把命令放到 Unity 的渲染佇列上。這些命令將在 Unity 的渲染執行緒上執行。為了設置各種渲染參數(如 GlyphCacheParams),我們還保留我們自己的共用佇列,並在從 Unity 內調用「顯示」(Display) 時處理該共用佇列。此佇列使用 Platform(平臺)專案(標準 Scaleform SDK 的一部分)中提供的實現方法。

另外,如 Unity 說明文檔所述,渲染僅在 PC(而非 iOS/Android)上具有多執行緒特性。此集成同時支援多執行緒渲染和單線程渲染。

8.1.1 使用命名空间

所有與 Scaleform 集成相關的 C# 代碼均包含在 Scaleform 或 Scaleform::GFx 命名空間中。這與 Scaleform SDK 中使用的命名空間和類命名約定相一致。例如,Movie(電影)類就是同時用 C++ 和 C# 在 Scaleform::GFx 命名空間中定義的。Scaleform 命名空間中包含 C++ 中沒有副本的 C# 類(如 SFManager)。這有助於避免與可能具有相同名稱的 Unity 類發生衝突。

8.1.2 创建和销毁 Scaleform 运行时

在我們的集成中,有三個初始化步驟。首先是創建 Scaleform 運行時、FManager、Scaleform 渲染器和硬體抽象層 (HAL)。

在 PC 上初始化 Scaleform 運行時的操作是在 UnitySetGraphicsDevice(從 Unity 內調用)中完成的。假如 D3D 渲染器正在使用當中,在此函數中,也會將指向 D3D 設備的指標作為一個參數進行傳遞。在 iOS/Android 上,在 SF_Init 中執行 Scaleform 運行時初始化。

第二個步驟是在 Scaleform 載入程式 (Scaleform Loader) 上設置各種狀態以及將 InitParams 結構 (SFCamera.cs) 中指定的設置傳遞到 Scaleform 運行時,這樣,就可按照使用者指定的設置初始化運行時。此操作是在 SF_Init 函數(從 SFCamera:Start 中的 Script 那裡調用)完成的。

第三個步驟是初始化第一步中創建的 HAL 物件。在多執行緒系統中,HAL 初始化必須在渲染器執行緒上發生。這是通過在 Windows 上發出一個 GL.IssuePluginEvent 調用和在 iOS/Android 上發出一個 UnityRenderEvent 調用 (eventId = 0) 來完成的。

SFManager 類包含該外掛程式的 C# 部分的大部分實現。必須在創建任何電影之前初始化此類。

可用兩種方法創建電影:通過調用 SFManager:CreateMovie,或者通過創建某個電影派生的類的新實例。兩種方法都將一個 SFMovieCreationParams 物件作為封裝電影名稱、視窗參數等的一個參數。一種典型的電影創建調用如下所示：

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

如下所述,電影是通過其 ID 標識的,而該 ID 只是與該電影對應的 C++ Movie 指標的整數表示。SFManager 在內部維護一個電影清單,用來處理事件、調用 Advance/Display(推進/顯示)等。

8.1.3 推进/显示

Flash 動畫是在 Movie:Advance 期間推進的。在此集成中,C++ Advance 是在 SFManager.Advance 期間被調用的,而後者又是在 SFCamera.Update 期間被調用 。電影是在 Display(顯示)期間調用的,而且,在 PC 上,由 Unity 通過 UnityRenderEvent 函數在渲染執行緒上調用 Display,而在 iOS/Android 上,則是通過調用 UnityRenderEvent (eventId = 1) 完成的。

Advance(推進)將上次更新以來過去的時間當作一個參數。這可用來控制推進電影的速度。假如您想要更加精細地控制某個電影的推進速度,可以覆蓋 Movie 派生的類中的 Advance 方法。

在多數常用使用方式下,Advance 的預設實現就足夠了。任何遊戲電影通信(如 invoke per tick)均應通過覆蓋 Movie.Update 函數(在 Advance 之前調用)進行實現。這使得遊戲更新與電影更新恰當地平行進行。

8.1.4 点击测试

點擊測試 (Hit-Testing) 用來檢查某個事件(例如,點擊滑鼠)是不是在 Flash 元素上發生的。點擊測試的結果可用來確定是否應將該事件傳遞到遊戲引擎。例如,如果您點擊一個 UI 按鈕,就可能需要防止遊戲進一步處理滑鼠點擊。此支援是通過 SFManager::DoHitTest 函數提供的。假如該輸入事件是在目前顯示的任何 Flash 電影上發生的,此函數就返回「真」(True)。

8.1.5 确定当前视窗

確定當前視窗的大小和位置在 Unity 中是一項極大的挑戰。Screen.Width/Height 屬性提供螢幕寬度/高度,然而,假如重新調整了視窗大小,或者更改了縱橫比 (Aspect Ratio),就不會正確更新這些屬性。就我們所知,沒有辦法在編輯器視窗中獲取視窗偏移量。為了克服此局限,我們在本機代碼中使用 OpenGL 視窗訪問函數,以便在調用顯示之前獲取視窗大小和位置。如果視窗發生變化,我們就將每個電影所使用的視窗重設為新的視窗尺寸。

可以使用 Movie.bAutoManageViewport 屬性來覆蓋此行為。如果將此屬性設置為「假」(False),就會始終使用(創建電影時設置的)預設視窗。
請注意,傳遞到 Scaleform 的所有事件的座標在內部自動轉換到當前視窗,而且不需要任何手動轉換。

8.1.6 从 ActionScript 调用 C# 函数

此集成使得聲明 C# 函數非常容易,而這些函數可以使用 ExternalInterface 從 ActionScript 進行調用。例如,假如您有一個包含電影剪輯的簡單 Flash 檔 Button.swf。您希望每次點擊一個按鈕時該電影剪輯的位置都根據播放機的位置而發生變化。此邏輯可實現如下:

用 ActionScript 定義一個滑鼠點擊處理常式:

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

```
};
```

現在在 C# 中,創建一個從 Movie 派生的類,並聲明一個將 Value(值)作為一個參數的函數「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。
	}
}
```

在內部,Unity 集成採用 C# 反射找出要在 ExternalInterface 回檔期間調用的正確的 C# 函數。為達此目的,您需要在一個 Movie 派生的類中聲明一個函數,該類具有與在 ExternalInterface.call 調用中使用的相同的簽名(相同名稱和參數)。如果找不到匹配的函數,ExternalInterface 調用就會默默地失敗。

8.1.7 事件处理

滑鼠和鍵盤事件被用 SFCamera:OnGui 發送到 Scaleform。每個 Movie 可以有自己的視窗,而且在將事件傳遞到 Scaleform 運行時之前,每個電影在內部執行滑鼠位置到電影視窗的轉換。通過覆蓋 Movie.HandleMouseEvent 可以更改預設轉換。您也可以通過覆蓋 AcceptMouse/CharEvents 並返回「假」(False) 來預防該電影不回應 Mouse/Key 事件。

8.1.8 在脚本中表示电影

使用一個整數 ID 在腳本中表示電影。此 ID 與 C++ 中的 Movie 指標所表示的整數值相對應。此技術使得通過簡單地將 Movie 物件歸於一個 Movie 指標來標識用一個 ID 表示的 Movie 物件。

```
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# 創建的物件的壽命是自動管理的。C# 運行時定期調用垃圾收集器 (Garbage Collector),以便於銷毀那些不具有任何活動引用的物件。Movie 類和 Value 類分別用來表示 Scaleform 電影和 Scaleform 值。

因此,C++ 電影和值的壽命與相應的 C# 物件的壽命密切相關。為了確保 Scaleform 電影和值正確銷毀,我們覆蓋了相應的 C# 類的 Finalize 方法,並手動銷毀 C++ 物件。

需要注意的另外一點是,C# 垃圾收集器運行于一個與主遊戲執行緒不同的執行緒。因此,必須確保調用 C++ Values 和 Movie 方法時執行緒安全。

8.1.10 Direct Access API

Direct Access API (DAPI) 用來直接存取和修改 Flash 物件及其屬性。DAPI 消除了用 ActionScript 創建函數並在每次需要訪問 Flash 物件的某個屬性時調用它們的必要,因而極大地提高了速度和效率。Scaleform 使用 Value(值)介面表示簡單類型(如 Int、Bool)以及複雜類型(如 DisplayObject)。Unity Integration 為整個 DAPI 介面提供一個包裝層,這樣,使用者就可以直接存取用 C# 編寫的 Flash 物件。DAPI 使您能夠用 C# 編寫大多數 UI 邏輯,而不必編寫許多 ActionScript 代碼。有關更多資訊,請參閱 Value.cs 並查閱 StarShip 演示中的 Hud 和「暫停功能表」(Pause Menu) 的實現。

8.1.11 跟踪语句

Actionscript 跟蹤語句被自動定向到 Unity Console 輸出,以便於調試。

8.1.12 部署

在我們的演示中,Flash 素材位於 Assets/StreamingAssets 資料夾。無需修改,此資料夾中的檔會被自動複製到部署平臺。

9 iOS 上的特殊考虑因素

在 iOS 平臺上進行的開發和部署存在不少差異。本節概述其中許多差異。

9.1.1 的行为

如上所述,pinvoke 用於託管代碼與非託管代碼之間的通信。在 iOS 與 Windows 平臺上實現 pinvoke,二者之間存在明顯差異。接下來我們將重點介紹其中幾項差異：

1. 在 iOS 上,「代理」(Delegates) 無效:如前所述,在 Windows 上,我們使用代理來從本機代碼調用託管代碼方法。使用代理是一種方便,因為它們立即執行並可傳回值。令人遺憾的是,由於在 iOS Mono 上 AOT(提前)編譯產生的局限性,使得代理不工作。為了克服這一局限性,我們將外部介面通知和值參數放到一個共用佇列上。此佇列按幀輪詢,而且就像常規 ExternalInterface 回檔一樣,處理該佇列上存儲的任何命令。不過,對 ExternalInterface 回檔進行排隊意味著不再立即執行這些回檔,而是在 Unity 的更新期間進行(有關實現的詳細情況,請參閱 SFManager::ProcessCommands()),這使得這些回檔不同步。此外,回檔無法傳回值。我們發現這些局限帶來不便,但並不構成多大問題。

2. Marshal.PtrToStructure 方法無效。這意味著,必須手動完成將資料從非託管堆指標封送到類的操作。從使用者的觀點看,此局限並不十分重要。

10 渲染到紋理

「渲染到紋理」是一種重要的電腦圖形技術，可用來創建許多令人關注且賞心悅目的效果。此技術使您能夠將您的 Flash 內容拖到一個遊戲紋理(而非背景緩衝區)。然後可以將此紋理應用到您的場景中的任意 3D 物件。下面的截屏顯示一些 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。

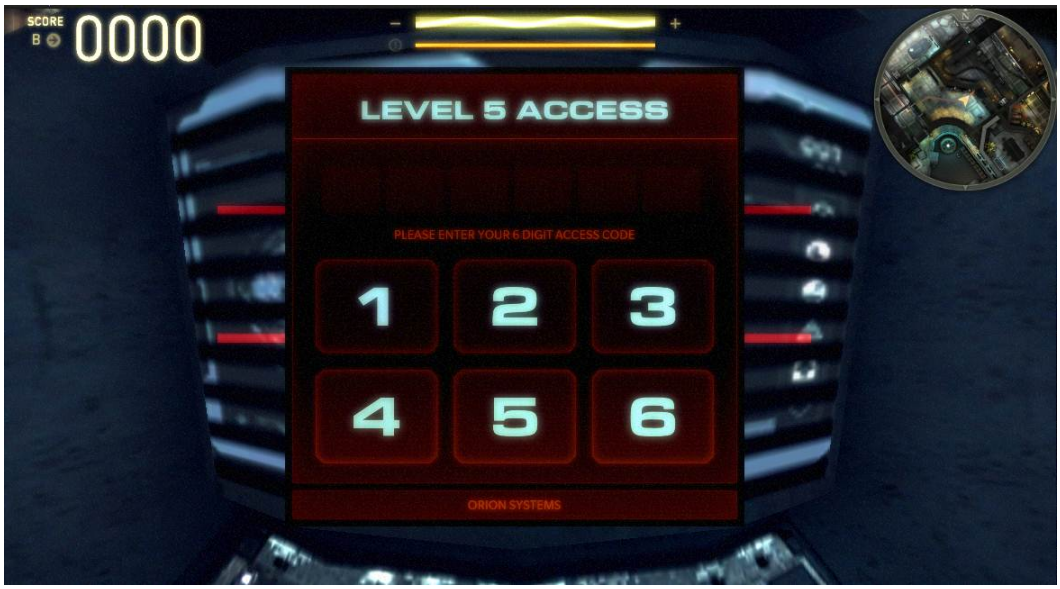


图 3：渲染纹理的另一个示例。该 Flash 电影可以接受用来点击键盘按钮的鼠标输入

10.1 在 Unity 集成中使用渲染纹理

Scaleform-Unity 集成使將您的 Flash 電影設置為渲染到紋理(而不是背景緩衝區)變得非常簡單。請按照下面的步驟渲染到紋理。

1. 生成 SFRTT 子類(Plugins\SF\SFRTT.cs 中定義),並添加用來創建 RenderTexture 電影的代碼。您可以跟隨 HelloWorldDemo 中的 MyRTT.cs 中的代碼去看一個示例。如果您的 Flash 電影不發送任何外部介面回檔,您可以直接使用 Movie 類,而無需進行子類生成。

2. 如果想要您的渲染紋理電影回應 Flash 事件,如滑鼠點擊,您就必須在創建 RenderTexture (RTT) 電影的同時覆蓋該 Movie 類,並提供子類的名稱。這與針對疊加電影的設置相同。

3. 在 Unity 編輯器中,將您在步驟 1 中創建的派生類拖放到您要在其上面繪製紋理的物件。

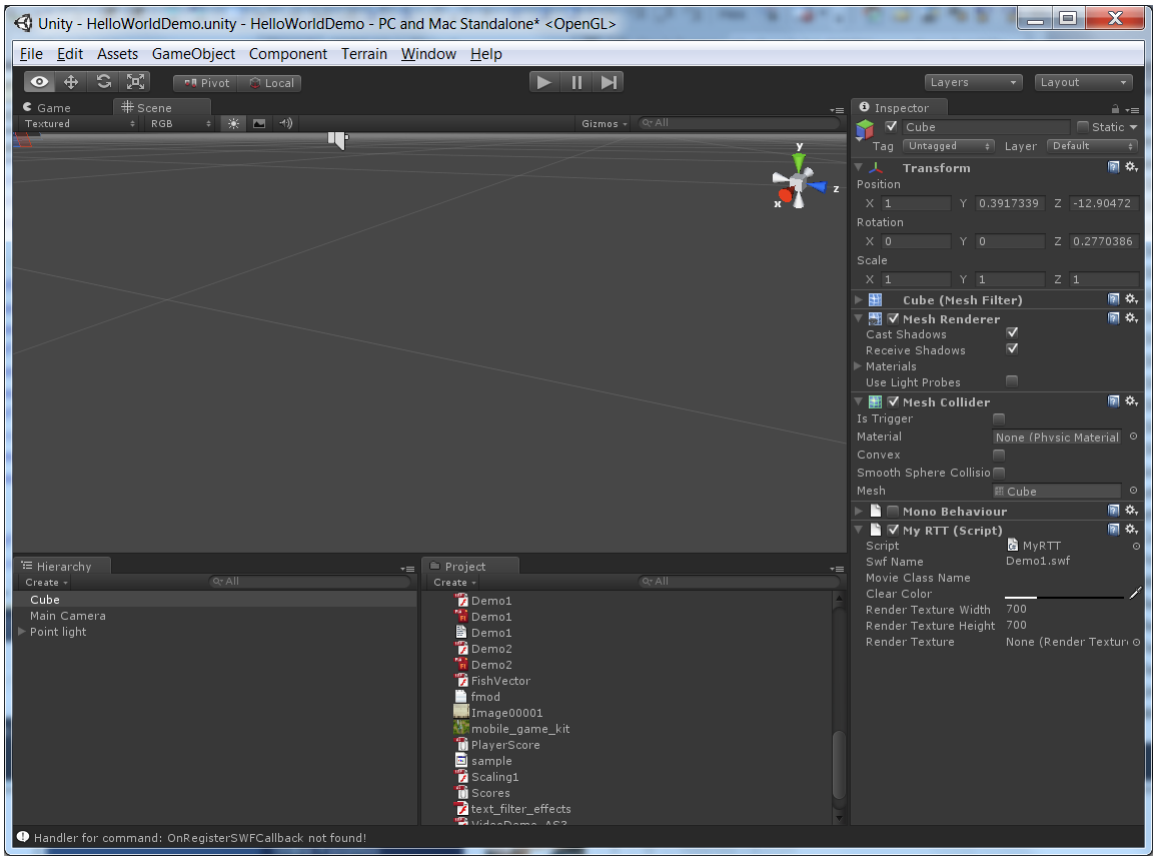


图 4：HelloWorldDemo 中的 RenderTexture 设置

您可以指定實現外部介面回檔的 Movie 子類名、Flash 電影的名稱(不要忘記 .swf)以及編輯器本身內的渲染紋理的寬度和高度。重要提示:確保寬度和高度相同。否則 RenderToTexture 就不發揮作用。這似乎是 Unity 的一個局限。您也可以為編輯器中的目標紋理指定清晰顏色。

10.2 點擊測試

我們實現了將滑鼠事件轉換為 RTT 電影座標空間的邏輯(SFMovie.cs 中的 HandleMouseEvents)。此邏輯使用 MeshCollider 元件計算發生滑鼠點擊的紋理空間中的點。為此,使用 RTT 的 3D 物件必須附加有一個 MeshCollider 元件。

此邏輯旨在證明進行點擊測試 (Hit Test) 的一種方法。如有必要,您可以添加自己的自訂點擊測試邏輯。從內部來看,Scaleform 僅使用從 C# 傳遞來的座標。

10.3 將焦點轉移到 RTT 電影

SWF 電影需要用來接收使用者輸入的焦點。電影並不總是能夠輕而易舉地接收輸入,當渲染到一個看不見的紋理時尤其如此。Scaleform 使得控制特定電影是否接收來自使用者的輸入成為可能。這可以通過切換電影的焦點(SFMovie.cs 中的 SetFocus)來實現。

遊戲程式師完全可以確定渲染紋理電影何時適合獲得或失去焦點。可以接受點擊測試的紋理非常適合接收滑鼠輸入。換句話說,鍵盤輸入並不具有此種直接包含測試。

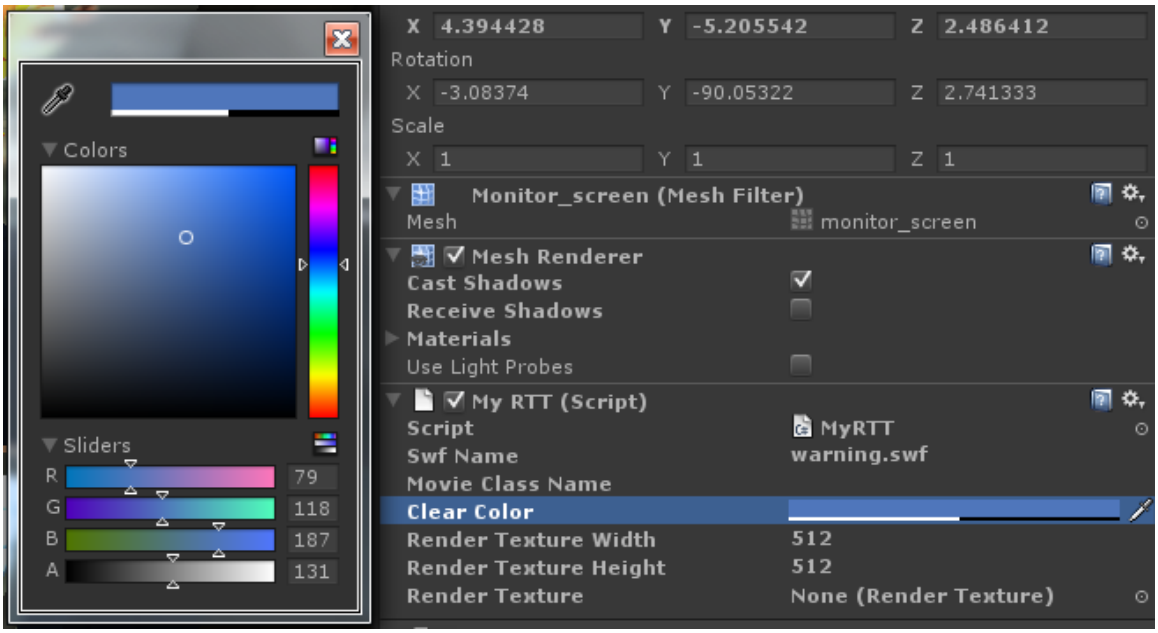
確定是否適合將使用者鍵盤輸入發送到渲染紋理的方法有多種。例如,可以通過測試來瞭解某個物件是否在視圖錐截體 (Viewing Frustum) 內。在某些情況下,場景中可以同時存在多個渲染紋理,因而將鍵盤事件發送到所有可見電影可能並不具有什麼意義。因此,在 Starship Down 中,我們選擇根據我們的玩家與上述每個渲染紋理之間的距離遠近來設定焦點。

10.4 添加阿爾法混合

10.4.1 透明電影

除了 Scaleform 中提供的正常「渲染到紋理」功能之外,還可以毫不費力地擁有阿爾法混合 (Alpha-blended) 的電影。下面的步驟將介紹如何達到此目的：

1. 使用本文前面介紹的指導說明創建一個渲染到紋理物件。
2. **確保該物件的「清晰顏色」(Clear Color) 參數擁有一個不小於 255 的阿爾法值。例如,在下面的圖像中,將用一種淡藍色背景渲染 SWF,該背景具有大約 50% 的透明度。**



3. 確保指定給該物件的材料是透明的。例如,可能使用「手機」（Mobile）-> 「透明」（Transparent）類別中的「頂點顏色」(Vertex Color) 材料：

4. Continue enjoying the Scaleform plugin for Unity.

11 附录

11.1 Interop

託管代碼和非託管代碼通過平臺介面服務 (PInvoke) 系統彼此互動。在 Windows 上,系統工作方式如下：

假定您要從 C# 調用函數 C++ foo(char*)。首先,在一個 .cpp 檔中實現此函數,並用一個 dll 將其匯出。

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

然後,在非託管代碼 (C#) 中,使用 DllImport 屬性聲明此函數：

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

現在,就可以就像一個常規函數一樣,從非託管代碼調用此函數:foo("Hello World");

請注意,pInvoke 封送層負責處理從 C# String 到 C++ char* 的轉換。pInvoke 自動實現單一資料型別的封送,但較複雜的類型可能必須手動封送。例如,DisplayInfo 結構,它封裝 Flash 顯示物件的顯示內容。有關封送和 pinvoke 的詳細資訊,請參閱 MSDN 說明文檔。

現在,我們看看轉換情況- 從非託管代碼調用託管代碼中的函數。要想調用腳本函數以回應 ExternalInterface 回檔,就需要執行此操作。在 Windows 上,這是通過代理實現的。

託管代碼：

```
// 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

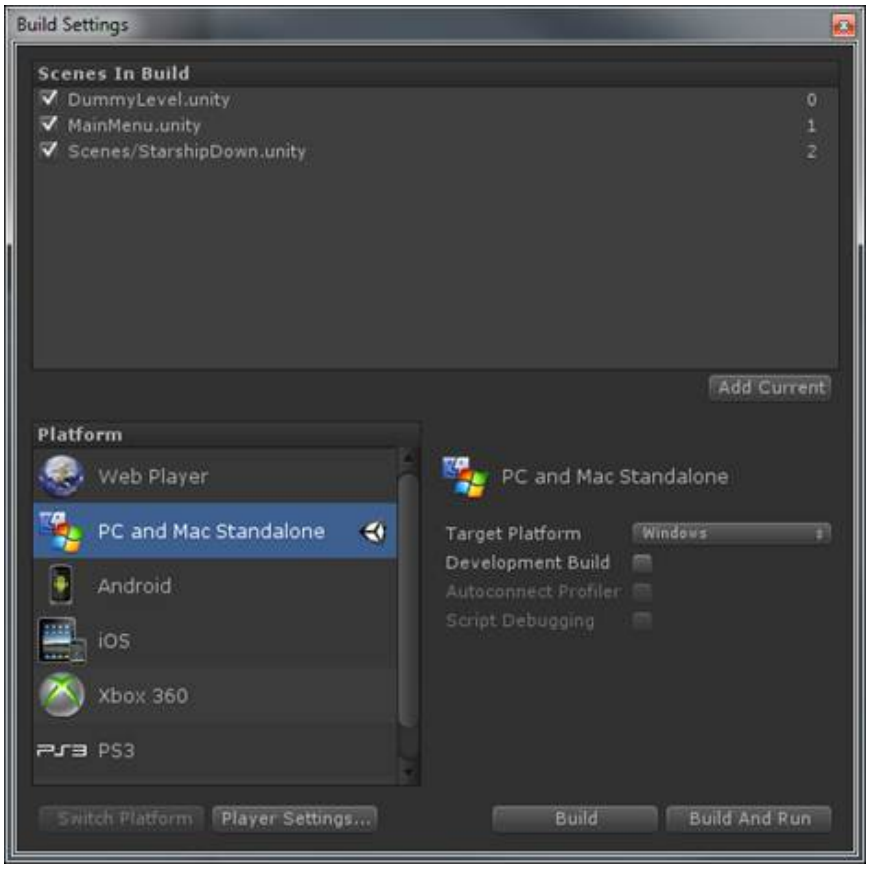
要在 Windows 平臺上構建和調試 Starship 演示,請遵循下述指導說明。下一節介紹 iOS 的構建說明。

11.2.1 在不调试的情况下运行演示

1. 從您安裝 Unity 的命令提示處(一般為 C:\Program Files (x86)\Unity\Editor)啟動 Unity.exe。預設情況下,使用 D3D 渲染器啟動 Unity 編輯器。如果想要使用 OpenGL 渲染器,請用命令列選項「-force-opengl」 啟動 Unity.exe。請注意,在 PC 上,HelloworldDemo 和 StarshipDown 中包含的預設 dll 是發佈-D3D,因此,要想使用 OpenGL 渲染,就必須從 Unity/Bin 複製適當的 OpenGL dll。

2. 從檔功能表上,流覽到 Integrations\Unity\StarshipDown\Assets 並打開 DummyLevel.Unity。

3. 現在,在「構建設置」(Build Settings)(「檔」(File) ->「構建設置」(Build Settings))中,如下圖所示,給 DummyLevel、MainMenu 和 StarshipDown 指定關卡級數:

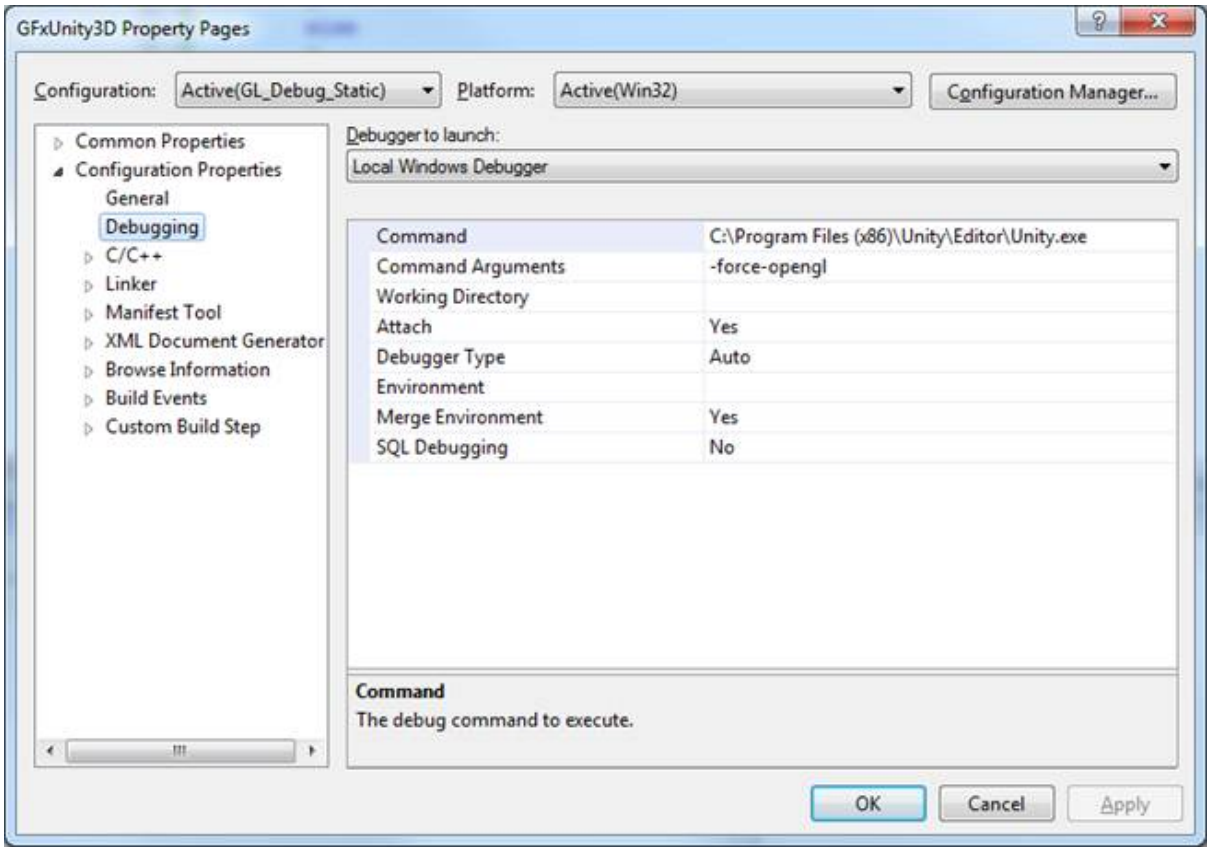


4. 按「播放」(Play)。現在,應該會看到 Scaleform 主功能表。

11.2.2 使用 **Visual Studio** 进行调试

注意:只有可以訪問 Src 的客戶才能使用此集成的原始程式碼。假如您想要獲得原始程式碼存取權限,請寫信給 Scaleform 支援部門。如果您不是源客戶,本節與您無關。

從 Integrations\Unity\Projects\Win32\Msvc90 打開 GFxUnity3D,並如下圖所示更改「配置屬性」(Configuration Properties)：



現在按 Cntrl+F5。這應啟動 Unity。請注意,假如您以前沒有用 DummyLevel 啟動 Unity,就不會是 Unity 啟動期間啟動的預設關卡,而且必須流覽到手動查找 DummyLevel 的資料夾(如上所述)。

要連接到偵錯工具,請按 F5,現在您應該能夠在集成代碼中設置中斷點了。

11.3在 iOS 上构建

在 iOS 上構建 Unity 應用程式的過程與在 Windows 上有很大差異。最大的差別是 Scaleform 外掛程式實際連結到您的應用程式,而不是用作一個 dll。因此,必須用 DlliImport(「__Internal」) 代替 C# 檔中的 DllImport(「libgfxunity3d」) 語句。為使此操作更加容易和透明,我們將對使用導入的函數的類的實現拆分為兩個檔。例如,把對 Movie(電影)的實現拆分成 SFMovie.cs 和 SFMovie_Imports.cs。Imports 檔包含 ifdef's,這樣,就會自動使用版本正確的導入函數。

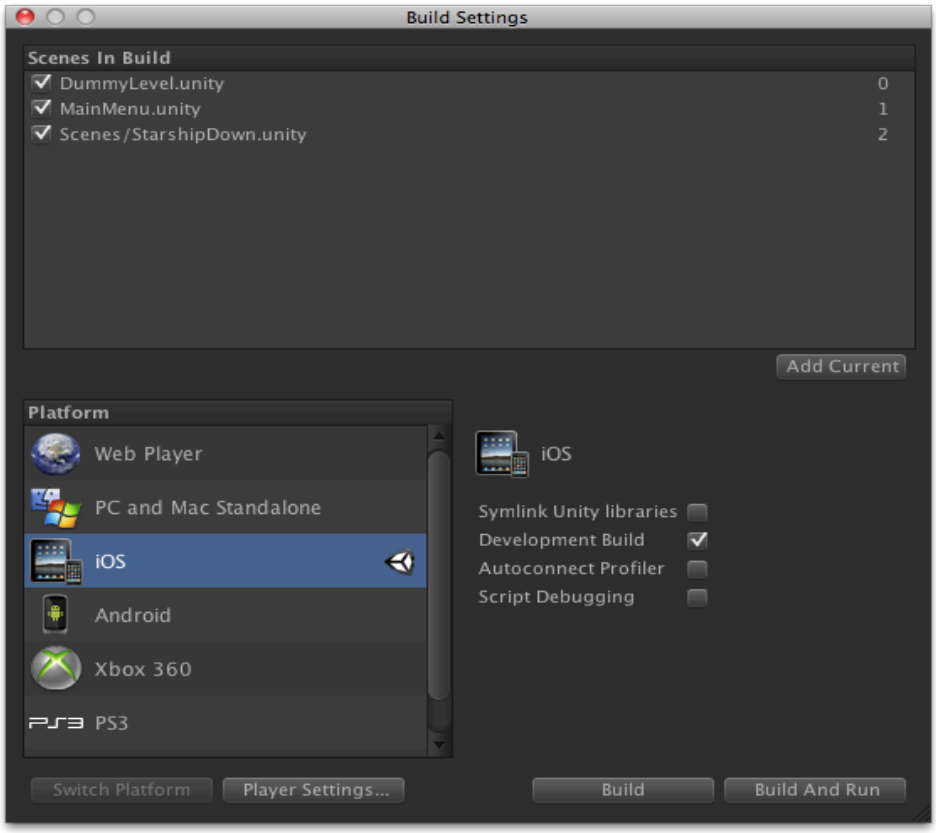
團結生成此演示的 Xcode 專案，然後通過添加集成代碼和 Scaleform 庫在 Xcode 內進行修改，以支援 Scaleform。下面的路徑下的集成套裝程式中包含有用於飛船下降演示的 iOS Xcode 專案的一個預構建副本：

- StarshipDown/iOS Project/

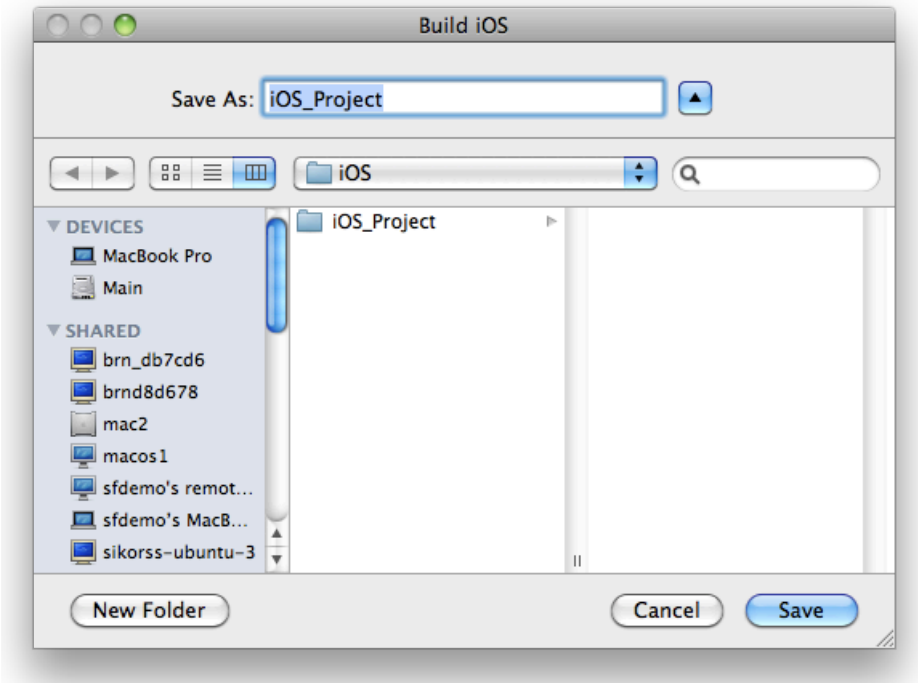
此 StarshipDown 專案為客戶提供一個參考,用來在 Scaleform 支援的情況下配置自己的 Unity Xcode 專案。為達此目的,將集成代碼、Scaleform 庫、素材等添加到 Xcode 專案,並更新了該專案的「專案設置」(Project Settings)。這些都有助於有效地將此集成編譯到您的遊戲應用程式之中,而且是 iOS 所必需的過程。

無論任何 Unity 內容何時發生變化(「場景」(Scene)、「網格」(Mesh)、「材料」(Material)、「Unity 腳本」(Unity Script) 等),在 iOS 設備上查看 Xcode 專案之前,都需要對其進行更新。您可以使用下面的步驟更新該專案：

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



2. 確保「構建中的場景」(Scenes In Build) 中列出的「場景」(Scenes) 正確,且排列順序正確。
3. 在「平臺」(Platform) 下,選擇 iOS。
4. 按一下「構建」(Build) 按鈕。
5. 提示保存檔的位置時,選擇適當的位置(一般情況下,該位置是一個現有 Xcode 專案的位置)。



6. 提示是要「附加」(Append)、「替換」(Replace) 還是「取消」(Cancel) 時,選擇「附加」(Append)。請注意,選擇「替換」(Replace) 將會導致重新構建整個 Xcode 專案,刪除可以配置的任何 Xcode 「專案設置」(Project Settings)。

7. 打開 Xcode 專案。
8. 從 Xcode 構建並運行/調試該專案。

假如這是您首次在 iOS 上運行「專案」(Project),請注意,您必須在 iOS 的「Unity 專案設置」(Unity Project Settings) 內安裝一個有效的「供給設定檔」(Provisioning Profile) 和「捆綁識別碼」(Bundle Identifier)。有關正確配置供給設定檔和捆綁識別碼的更多資訊,請訪問 Unity 的網站和 Apple 開發者支援網站。

要配置自己的 Xcode 專案以支援 Scaleform 4.1,請確保使用下列設置：

Unity 播放器设置 ：

目標平臺：armv7

SDK 版本：iOS 5.x or 6.x

目標 iOS 版本： 5.x or 6.x

這很重要,否則您在 iOS 上部署應用程式時就可能會收到下面的錯誤：

```
"Uncaught Exception:
*** -[PBXDebugScriptCommand debugSessionDidStart:]: unrecognized selector sent to
instance 0x49726c0"
```

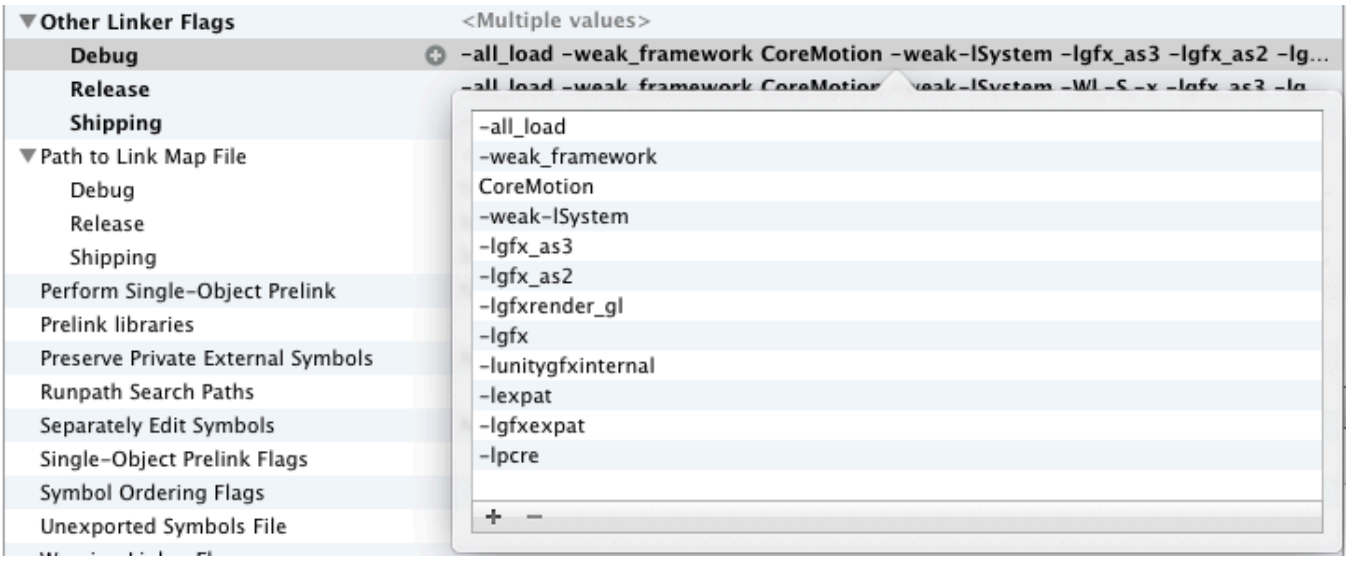
（“未捕获到异常：

*** -[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 外掛程式代碼：
    - {SDK}/Integrations/Unity/Src/SFExports.cpp

- Scaleform 庫(針對各種配置,將這些添加到您的 Target 的構造設置中 – 我們將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 檔(例如,UI_HUD.swf、UI_LoadingScreen.swf 等)應包含在您的專案的 StreamingAssets 資料夾中,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

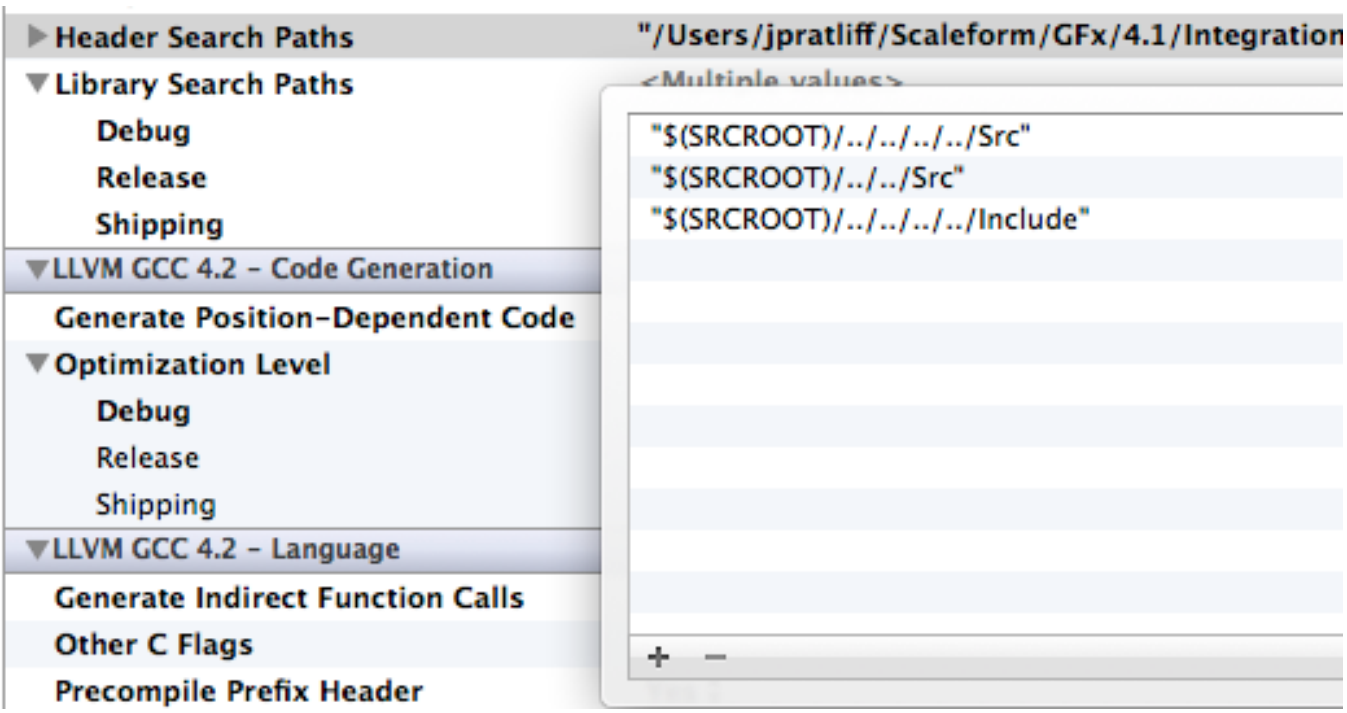
代码生成：

縮略圖編譯：假 (False)

搜索路径：

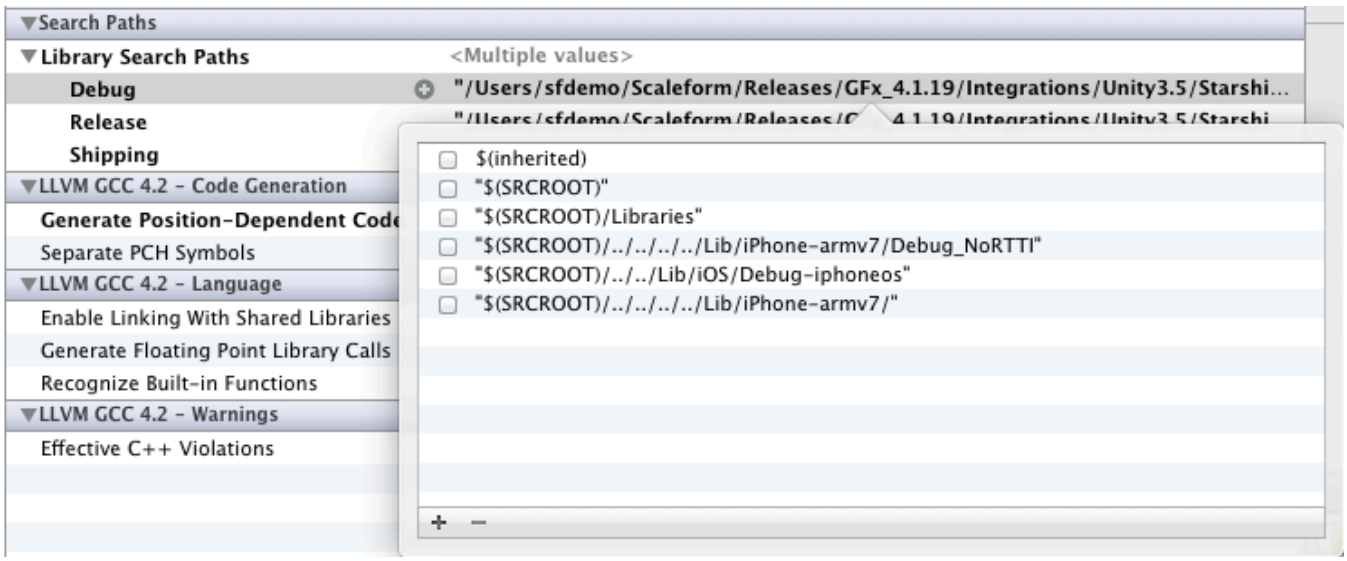
標題搜索路徑：

- {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：真 (True)

启用 C++ 运行时类型：假 (False)

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 –预处理

預設情況下,Unity 不在針對 iOS 的 OpenGLES2 實現中提供一個範本附件 (Stencil Attachment)。這可防止 Scaleform 繪製遮罩。要添加遮罩支援,您需要在 iPhone_GlesSupport.cpp ({SDK}/Integrations/Unity/StarshipDown/iOS_Project/Classes/) 的第 145 行進行如下代碼更改。請注意,更改以黃色突出顯示：

```
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 上運行您的支援 Scaleform 的 Unity 應用程式與在 Windows 相似。不需要編譯任何 No C++ 代碼。如前所述,Scaleform 運行時包含在位於 Unity/Bin/Android 的 libgfxunity3d.so 共用物件檔中。為了方便起見,這些檔也複製到 HelloWorldDemo 和 StarshipDown 演示的 Assets/Plugins/Android 目錄。

11.4.1 运行 HelloWorldDemo

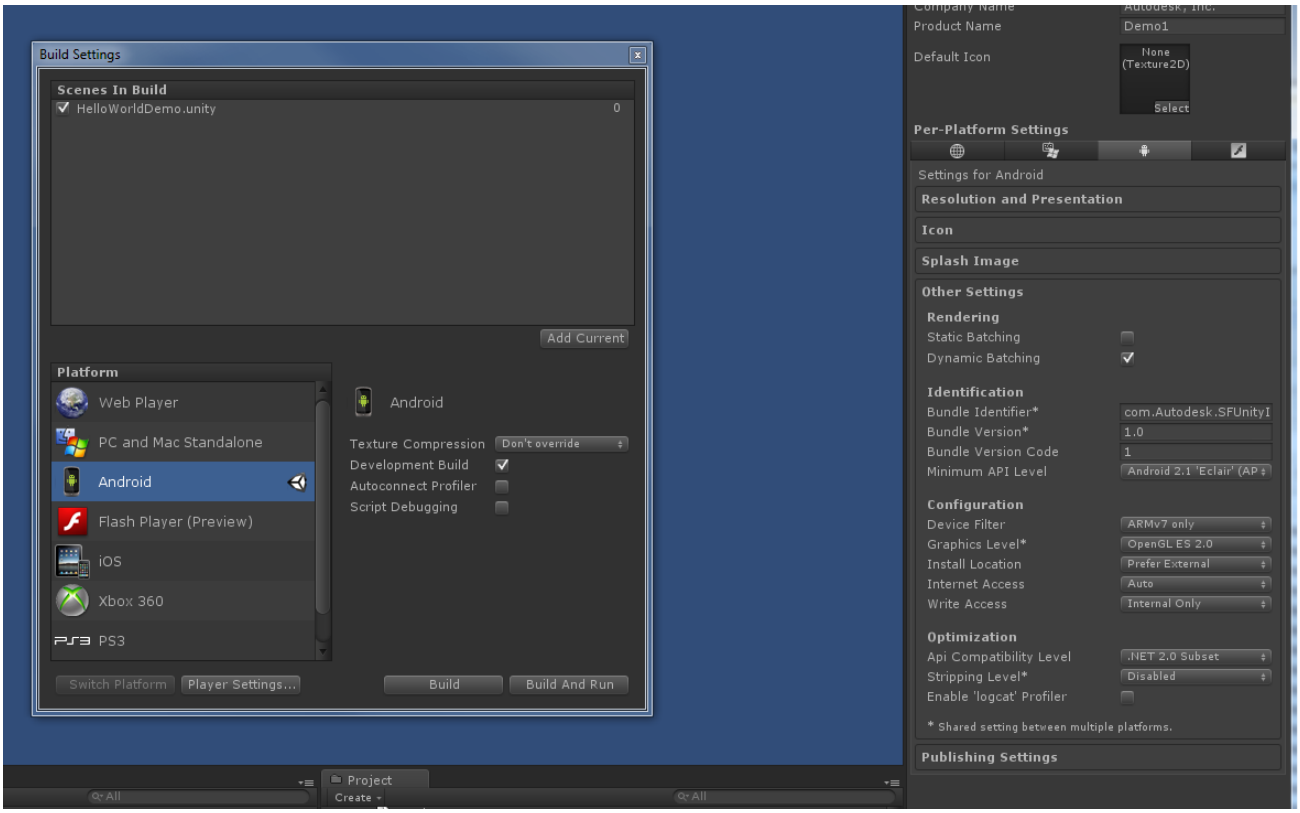
打開 Unity 編輯器,並流覽到 HelloWorldDemo/Assets/。打開 HelloWorldDemo.unity 並轉到構建設置,並選擇 Android 作為目標平臺。 點擊「構建和運行」(Build and Run)。如果一切設置正確(請查閱下面有關播放機設置的說明),Unity 應構建 HelloWorldDemo.apk 並在設備上啟動該程式。

11.4.2 创建自己的应用程序

創建 Android 應用程式的關鍵步驟是：

1. 將核心集成腳本檔案複製到 Assets\Plugins\SF 中
2. 創建一個覆蓋 SFCamera 的腳本,並將其附加到您的場景中的一個相機物件
3. 將 libgfxunity3d.so 複製到 Assets\Plugins\Android 資料夾
4. 確保您的 Flash 素材位於 Assets\StreamingAssets

現在,在 Unity 編輯器中,切換到 Unity 平臺,並確保按下圖所示設置播放機設置。



重要設置：

設備篩檢程式 (Device Filter): ARMv7
圖形等級(Graphics Level): OpenGL ES 2.0
最低 API 等級 (Minimum API Level): API level 7

11.4.3 使用 cygwin 在 Android 设备上启动 .apk

第 1 步:流覽到 .apk 所在的目錄。
第 2 步:在 Cygwin 命令視窗上,鍵入：`adb install HelloWorldDemo.apk`
第 3 步:要看到調試日誌,請使用：`$ adb logcat -s Unity ActivityManager PackageManager dalvikvm DEBUG`