Autodesk®
Scaleform®

Scaleform 視頻入門

本文檔詳細描述了在 Scaleform 4.3 中 Scaleform 視頻的使用

作者： Matthew Doyle, Vladislav Merker
版本： 3.01
上次修訂：September 15, 2011

Autodesk®
GAMEWARE

Copyright Notice

Autodesk® Scaleform® 4.3

Autodesk Scaleform 聯繫方式：

文檔	Scaleform 視頻入門
地址	Scaleform Corporation 6305 Ivy Lane, Suite 310 Greenbelt, MD 20770, USA
網站	www.scaleform.com
郵箱	info@scaleform.com
電話	(301) 446-3200
傳真	(301) 446-3199

目錄

1 歡迎

Scaleform® Video™由 by CRI™提供，是一個集成到 Scaleform™的完整視頻解決方案，是一個插件模組允許用戶爲 Adobe® Flash®增加高性能視頻播放功能，充分利用了集成的 Flash 視頻管道的優勢。使用此新的視頻模組，開發者可以播放高解析度、高清晰度的畫面，如標誌、主功能表、HUD、遊戲內繪製以及全屏影片場景切割。

CRI 視頻編碼器根據播放功能和相對於當前視頻編碼器的編碼優勢進行選擇。CRI 視頻的下一代播放引擎轉爲即時遊戲系統構建，利用了最新多核硬體的優勢。CRI 中間件已經在超過 1700 個遊戲中獲得應用。

Scaleform 視頻允許視頻在任何解析度下播放，支援 PC 上和遊戲控制器(Xbox 360®、PLAYSTATION®3"(PS3™)、Wii™)，全屏模式，表單內，Alpha 透明效果以及更多其他功能。完全設計成流水線和多線程方式。使用 Scaleform 視頻可以爲不同的平臺導出不同解析度的視頻以優化質量、幀速率、解析度、比特率、尺寸、螢幕高寬比以及更多平臺相關功能，確保視頻質量和性能的優化。

Scaleform 視頻工作流使得用戶可以簡單得在 Adobe Premiere®、Adobe After Effects®或者其他任何視頻編輯應用工具中創建視頻文件，可以將其導入到 Adobe Flash 並快速簡單得導出到遊戲中。在處理過程中，可以方便得創建標題、添加 5.1 聲道音效和選擇不同語言配音，交互小插件使遊戲內視頻交互功能更強。Scaleform 視頻同樣還有其他一些高級特性，如 alpha 通道、播放暗點，這些特性允許開發者利用視頻功能不僅僅只是介面和場景分割。

Scaleform Video SDK 包括以下項：

- Scaleform 視頻即時運行庫
- Scaleform 視頻工具
- 指南/實例
- Scaleform 視頻入門(本文檔)

2 關於 Scaleform 視頻

Scaleform 視頻是一種高解析度、高聲音質量視頻播放系統，支援多平臺應用。用來幫助創建符合全球最高標準，充分利用每個平臺的特性的高質量視頻。同時也具有高級特性，如多語種配音和字幕，遊戲可以在不同的國家和地區銷售。

2.1 安裝位置

Scaleform 視頻安裝視頻編碼器位置爲：

C:\Program Files\Scaleform\GFx SDK 4.3\Bin\Tools\VideoEncoder

Scaleform 視頻安裝一個視頻演示文件到以下位置：

C:\Program Files\Scaleform\GFx SDK 4.3\Bin\Data\AS2\Video

本演示包括一個包含三個不同視頻播放器的視窗，可以獨立播放視頻文件。本目錄也包含一些演示文件相關的 Flash 文件和視頻文件，包括一些預編碼 USM 文件用來做測試用。在 Scaleform 播放器中運行 *videodemo.swf* 來試播演示文件。

2.2 指導文件

在編碼器軟體相同的位置，還有一些例子文件配合本指導文件使用。使用這些例子需要遵照指導文件中所描述的步驟。

- *getting_started_with_video_tutorial.fla* – 本 Flash 文件包含了本指導手冊列出的所有內容。
- *getting_started_with_video_tutorial.swf* – 這是 Flash 文件的發佈版本。
- *sample.avi* – 編碼器的視頻採樣。
- *sample_audio.wav* – 編碼器的音頻採樣。
- *sample_cue_points.txt* – 編碼器的採樣點。
- *sample_subtitles.txt* – 編碼器的採樣字幕。

2.3 編碼器目錄

在 Scaleform 編碼器目錄下有一些操作中非常關鍵的文件。

- Medianoche.exe — 編碼器命令行。
- ScaleformVideoEncoder.exe — 編碼器結束之前的圖像(本文檔中有所描述)。
- TMPGLib.dll — 編碼器核心。
- VideoEncoderUtil.dll — ScaleformVideoEncoder.exe 必須要用的庫。
- VideoPlayer.swf — 當在 Scaleform 視頻編碼器視窗中點擊 *Preview* 時，這個即爲圖像視頻播放器，控制視頻文件輸出。

2.4 優勢

- **高質量視頻播放**
  高質量視頻能夠使用簡單的 API 函數進行播放。動畫播放具有完全的高清晰度(HD)和解析度以及良好的效果（例如 1080p）。

- **支援多語言配音和字幕**
  多語言配音和字幕可以放在單個視頻文件中。爲不同地區銷售的遊戲減少的開發量。

- **專業級編碼簡單易用**
  SDK 包中包含了一個簡單易用的編碼工具。

2.5 特性

Scaleform®視頻，由 CRI Movie™提供，是一個插件模組，允許開發和通過 Adobe Flash®增加性能。使用 Scaleform 視頻可以在多樣化的下一代平臺中播放極高質量、高解析度和高清晰度視頻。可以播放視頻有：全屏、視窗、3D 遊戲引擎中透明背景。在不同應用的交互內容中利用視頻的主要包括：圖示、主功能表、遊戲 HUD、遊戲紋理、遊戲內視頻反映、全屏電影場景切割、導入畫面播放以及更多其他效果。在遊戲內部播放視頻不需要很多額外的工作，因爲 Scaleform 視頻已經完全集成進了 Scaleform™ 4.1。Scaleform 視頻構建在當前非常流行並聲譽卓著的 CRI Movie™編碼器之上，且顯著增強了其工作流並被 Adobe Flash 完全繼承。

2.6 技術指標

- **視頻和音頻格式:** 編碼器支援大多數常用的視頻和音頻格式，如 AVI 和 WAV。
- **多平臺支援:** Scaleform 視頻在多數主流平臺上都支援包括 Xbox 360、PS3、PC 和 Wii，具有硬體獨立性，可以自由編程。
- **不同平臺的編碼優化:** 在編碼過程中，可以應用優化參數，這些參數將平臺特性和視頻質量/壓縮率考慮在內。
- **字幕:** 同一個的影片支援多字幕通道。使得用戶可以爲不同的語言創建不同的字幕，存儲在單獨文件中，每種類型字幕都有不同的開始和結束時間進度。使用 Scaleform 字體配置系統多語言文本，能自動準確得定位。
- **Alpha 通道支援:** 用戶可以在包含內嵌 alpha 允許按圖元透明化以及整體透明化處理的動畫中創建“虛擬透明”效果。這種方法可以使用戶創造出多種不同且具有創意的效果。
- **紋理繪製:** 動畫能夠繪製在遊戲紋理中以及 3D 圖像表面，使得用戶可以直接將視頻集成到遊戲當中。
- **音效聲道:** 視頻支援多音效聲道（最多 32 個）。這使用戶可以擁有多語言配音。
- **聲道切換:** 用戶可以使用 using Scaleform Flash 4.1 擴展在 ActionScript™中不同聲道之間切換。
- **定位支援:** 在 Scaleform 視頻中通過 ActionScript 支援在運行時切換聲道和字幕。這是一項新的特性，使用不同語言的客戶可以更好的組織和管理 UI 視頻。
- **環繞聲支援:** 每個聲道可以爲單聲道/身歷聲或者 5.1 立體環繞聲。
- **多語言環繞聲:** 環繞聲中的中央通道資料可以被語音通道替代。這使用戶可以簡單得在環繞聲中播放語音聲道。
- **搜索能力:** 使用 Scaleform Video，用戶能在視頻流中搜索一個特殊的時間段內容。搜索定位值可以以視頻流開始爲參照或者當前位置爲參照。用戶能回退或者前進（使用正負搜索值）。
- **播放暗點:** 播放暗點爲視頻剪輯中重要的記號節點。播放暗點使得用戶可以訪問視頻剪輯中不同的片段。使用 Scaleform 視頻編碼器，能夠直接將播放暗點嵌入到 USM 文件。播放暗點也可以在 Flash 中使用。當到達視頻的播放暗點時，將觸發一個 ActionScript 事件，所有與該播放暗點相關聯的資訊傳遞給用戶。這允許創建導航功能表、搜索視頻中特殊事件以及使視頻產生交互效果。

2.7 播放文件

Scaleform 視頻將動畫轉換到爲 Scaleform CRI 視頻格式，文件尾碼名爲 USM。爲了播放 USM 視頻文件，只需將文件拖放到 Scaleform 媒體播放器圖示即可，該圖示應該在桌面上。也可以打開一個 Scaleform 播放器將 USM 文件拖放到打開的窗口中就能播放。

嘗試拖放 *C:\Program Files\Scaleform\GFx SDK 4.3\Bin\Data\AS2\Video\scaleform_logo.usm* 到播放器查看播放效果。

2.8 工作流

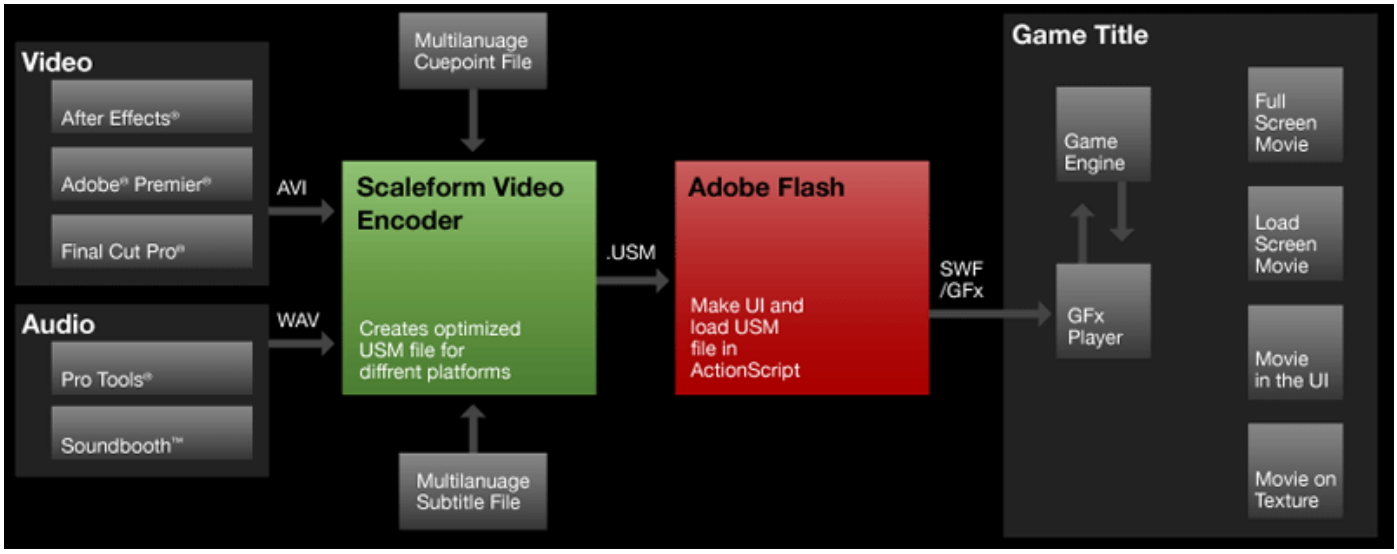

圖 1：Scaleform 視頻工作流

產生一個遊戲內部的視頻工作流如下所示：

1. 集成 Scaleform 視頻 API 函數到遊戲引擎.
2. 從 Adobe Premiere 或者其他視頻編碼器工具導出視頻爲 AVI 格式。
3. 使用 Scaleform 視頻編碼器轉換 AVI 視頻格式爲 USM 格式。
4. 創建一個 Adobe Flash SWF 文件包含 USM 編碼視頻。
5. 創建一些 ActionScript 腳本來控制視頻、字幕、播放暗點等。
6. 導入 SWF 文件到遊戲。
7. 使用遊戲引擎中可以的方法在遊戲中播放 SWF 文件。

***注釋：**不要試圖將 USM 視頻文件插入到 Flash 文件，可以通過 ActionScript 索引來代替。一旦創建了視頻並在遊戲中播放，視頻中需要重復使用，無需在相同的位置和以相同的檔案名重新對視頻進行編碼以更新任何 Flash 文件中內容。

3 Scaleform 視頻編碼入門

Scaleform™ 4.3 使用的視頻必須編碼成 CRI 的 USM 格式。CRI 格式爲需要高質量畫面和高解析度下實現高性的遊戲運行做了高度優化。爲便於使用，Scaleform 提供了 Scaleform 視頻編碼器。本章將一步步解釋如何使用編碼器對 USM 文件進行編碼，包括了對於播放暗點、字幕和聲音集成的詳細介紹。

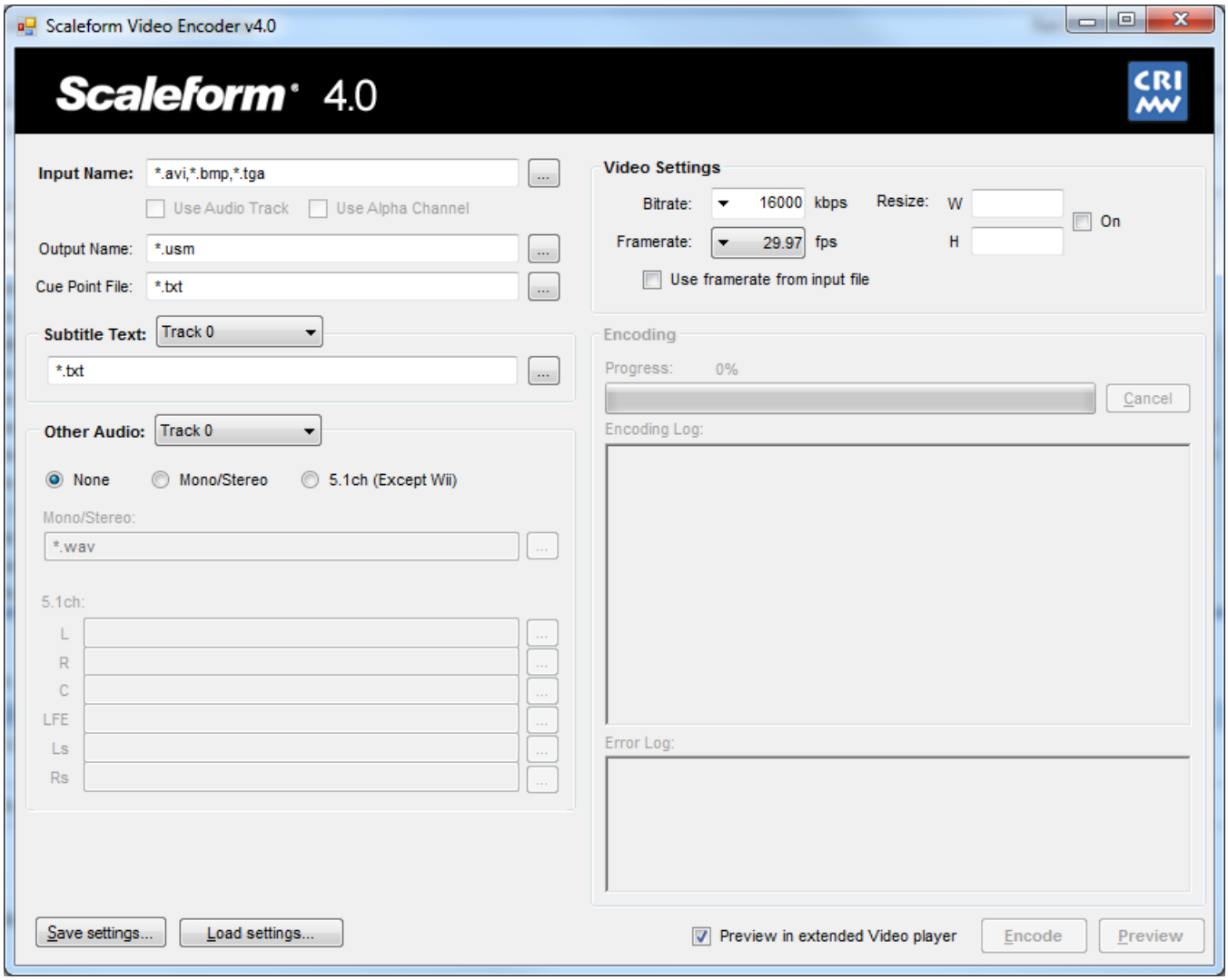

圖 2: Scaleform 視頻編碼器

3.1 基本編碼步驟

將視頻轉爲 USM 格式過程非常簡單。一些步驟，如字幕和播放暗點爲可選步驟。接下來將描述創建 USM 視頻文件所需的步驟，且包含了 alpha 通道（圖元透明）、播放暗點、字幕和身歷聲（2 通道）功能。

使用 Windows™開始功能表啓動 Scaleform 視頻編碼器。在 Scaleform 視頻默認安裝目錄下，可以在以下位置找到：
Windows Vista: *Start->All Programs->Scaleform->GFx SDK 4.3->Video*
Windows XP: *Start->Programs->Scaleform->GFx SDK 4.3->Video*

1. 使用 *Browse* 按鈕  來定位和導入 AVI 格式視頻文件進行轉換。
    a. 可能爲 AVI 視頻，或者 BMP 或 TGA 系列。
    b. 在 BMP 或 TGA 系列情況下，選擇序列中的第一個文件。

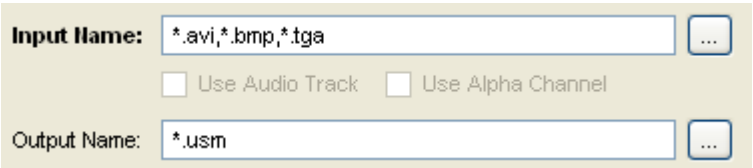

圖 3: 視頻文件編碼必須輸入名字

2. 如果視頻有內嵌聲道，則需要在 *Use Audio Track* 旁邊放一個標記。這個選項爲 AVI 默認提供的。在 BMP 和 TGA 序列中禁止使用。

3. 如果視頻具有 alpha 透明通道則標記 *Use Alpha Channel* 項將其編碼到最終 USM 視頻文件。這樣可以讓觀衆看到視頻之下 alpha 透明通道不是純白色地方的內容。此選項模式情況下爲未選中。

4. *Output Name* 域默認情況下爲源視頻路徑；但最好是通過 *browse* 按鈕瀏覽一個不同的目錄保存 USM 編碼文件。

5. 點擊 ‘*Encode*’ 按鈕啓動編碼。

3.2 第一個編碼舉例

1. 打開編碼器。
2. 導入文件 *C:\Program Files\Scaleform\GFx SDK 4.3\Bin\Data\Tools\VideoEncoder\sample.avi*。
3. 點擊 *Encode* to 開始以默認設置對示例動畫進行編碼。
4. 當編碼成功完成，*Preview* 按鈕有效。按下該按鈕預覽 *USM* 編碼視頻。

3.3 播放暗點

播放暗點對於設置章節導航非常有用，用來在 SWF 文件中改變資料或者關閉事件。如果視頻編碼未包含播放暗點，可以跳到下一章節；但是，建議閱讀播放暗點的簡要介紹。

爲了使最終的 USM 編碼文件使用播放暗點，點擊 *Cue Point File* 旁的 *Browse* 按鈕定位一個播放暗點文字檔案。

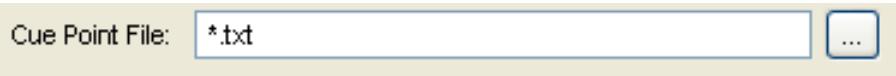

圖 4: 瀏覽播放暗點文字檔案。

一個播放暗點文字檔案包括每個播放暗點排列成的行。序列第一行應包含毫秒間隔顯示—典型爲 10000。這個間隔用來作爲決定每個播放暗點的時間值。一個時間值爲 5000 即爲 5 秒 — 5000/1000=5。間隔列表下的每行都有一個編碼視頻使用的單獨播放暗點。

播放暗點有兩種類型：導航和事件。導航播放暗點用來作爲章節選擇，很像 DVD 中的選段功能表。事件播放暗點用來設置參數。每個事件播放暗點可以有多個參數。參數列表爲鍵=鍵值。單個播放暗點的多個參數由逗號隔開。關於如何在 Flash 文件中使用播放暗點的詳細介紹，請參考 ActionScript 視頻應用。

單行播放暗點格式

```
time, cue point type (0 or 1), cue point name, parameter1, parameter2, …
paramter10
```

例子:

```
1000, 0, cue_point_1, my_param=5, my_other_param=10
```

CuePoint.txt 文件例子內容

```
1000
0,0,start
1000,0,cue1
2000,1,cue2,name1_0=value1_0,name1_1=value1_1
3000,0,cue3
4000,1,cue4,name4_0=value4_0
5000,0,cue5
6000,1,cue6,name6_0=value6_0,name6_1=value6_1
```

```
7000,0,cue7,name7_0=value7_0,name7_1=value7_1,name7_2=value7_2
8000,1,cue8,name8_0=value_0
9000,0,cue9
10000,1,cue10,name10_0=value10_0,name10_1=value10_1
11000,0,end
```

3.3.1 用播放暗點重新對示例視頻進行編碼

1. 打開 Scaleform 視頻編碼器。

2. 導入文件 *C:\Program Files\Scaleform\GFx SDK 4.3\Bin\Tools\VideoEncoder\sample.avi*

3. 點擊 *Cue Point File* 文本區域的 *browse* 按鈕，定位和導入 *sample_cue_points.txt* 文件 — 該文件位於 *sample.avi* 相同目錄中。

4. 點擊 *Encode* 用默認設置開始實例動畫編碼。

5. 當編碼過程順利結束，*Preview* 按鈕有效。點擊該按鈕預覽 USM 編碼視頻。

6. 在視頻播放器中使用 *fast forward* 和 *rewind* 按鈕進行播放暗點之間的快進和快退操作。

3.4 字幕

字幕文字檔案包括了視頻中的字幕行列表。字幕用來在視頻的特定時間顯示文本。如果視頻不包含字幕，可以跳過本節到下一個章節；但是，建議閱讀本節以理解如何添加字幕到視頻中。

允許多種字幕文件 — 每個文件爲一種字幕通道。這允許爲每種語言創建一個字幕文件，有些語言根據語言中字幕長度和閱讀所需時間需要不同的起始和結束時間。

1. 通過從 *Subtitle* 文本旁的下拉功能表中選擇字幕軌道的使用。最多支援 32 軌道 —Tracks 0-31。

2. 點擊 *browse* 選擇字幕文字檔案在軌道中使用。

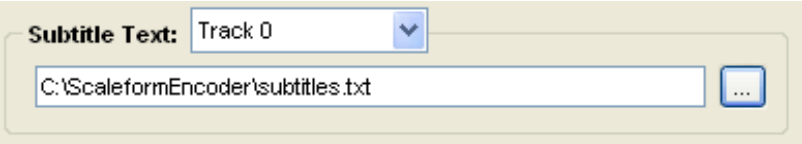

圖 5: 選擇字幕軌道並瀏覽文件。

字幕文件的第一行（不包括注釋行）應該包含以毫秒爲單位的顯示間隔時間（典型爲 1000）。這個值爲每個字幕的開始和結束時間的基準。在第一行後面包括三部分資訊：開始時間、結束時間、消息 ID。消息 ID 號在編碼中用來顯示文本字串相關資訊。同時，這個參數也可以被用來作爲顯示字幕的實際字串。關於如何在 Flash 文件中使用字幕，請參考 ActionScript 視頻應用。

單行字幕格式

```
start time, end time, messageID
```

Subtitle.txt 例子文件內容

```
1000
2000, 5000, This is the first subtitle.
8000, 11000, subtitleId2
```

3.4.1 視頻例子添加字幕重新編碼

1. 打開 Scaleform 視頻編碼器。

2. 導入文件 *C:\Program Files\Scaleform\GFx SDK 4.3 \Bin\Tools\VideoEncoder\sample.avi*

3. 點擊 *Cue Point File* 文本區域的 *browse* 按鈕，定位和導入 *sample_cue_points.txt* 文件 —該文件位於 *sample.avi* 相同目錄中。

4. 點擊 *Subtitle Text* 下拉功能表下面的文本區域旁的 browse 按鈕。定位並導入 *sample_subtitles.txt* —本文件位於 *sample.avi* 相同目錄中。

5. 點擊 *Encode* 用默認設置開始實例動畫編碼。

6. 當編碼過程成功完成後,從您的桌面快捷方式啟動 Scaleform Player (Scaleform 媒體播放機),然後將 *getting_started_with_video_tutorial.swf* 檔拖入其中。

7. 注意字幕顯示在視頻播放器播放視頻的底部。應該 2-5 秒產生一次，8-11 秒後再次產生。

3.5 音頻

編碼器的音頻通道允許添加多種音頻，包括單聲道/身歷聲或者 5.1 環繞聲。每個聲道可以包括不同的音頻文件，這可以用來進行多聲道立體環繞聲編碼。

1. 首先，在 Scaleform 視頻編碼器 *Other Audio* 旁邊的下拉功能表中選擇一個音頻軌道。

2. 選擇音頻類型 —無聲、單聲道/身歷聲或 5.1 聲道（除 Wii 之外）。
   注意：如名字所述，5.1 聲道在任天堂 Wii 中不可用；但是 5.1 聲道在其他任何平臺中都可以使用。

3. 對於單聲道/身歷聲，使用 *Mono/Stereo* 文本區域旁邊的 *Browse* 按鈕，定位音頻原始檔案爲 WAV 格式。

4. 只要 *Use Audio Track* 有效，本區域已指定到了視頻原始檔案所在路徑。

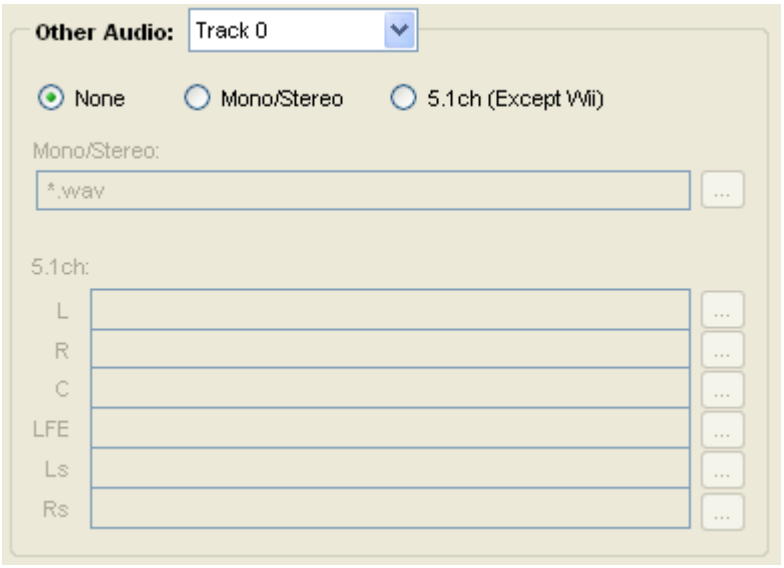

圖 6: 選擇字幕軌道並瀏覽查找一個文件

5. 對於 5.1 聲道音頻，使用每個通道列表旁的 *Browse* 按鈕 — *L*、*R*、*C*、*LFE*、*Ls*、*Rs* — 爲每個通道導入音頻原始檔案。

使用 5.1 聲道環繞聲的通道：
L – 左聲道
R – 右聲道
C – 中央聲道 (優先用於語音)
LFE – 重低音聲道

Ls –後置左環繞聲道
Rs – 後置右環繞聲道

3.5.1 採用雙聲道音頻軌道重新編碼視頻實例

1. 打開 Scaleform 視頻編碼器。

2. 導入文件 *C:\Program Files\Scaleform\GFx SDK 4.3\Bin\Tools\VideoEncoder\sample.avi*

3. 點擊 *Cue Point File* 文本區域的 *browse* 按鈕，定位和導入 *sample_cue_points.txt* 文件 — 該文件位於 *sample.avi* 相同目錄中。

4. 點擊 *Subtitle Text* 下拉功能表下面的文本區域旁的 browse 按鈕。定位並導入 *sample_subtitles.txt* — 本文件位於 *sample.avi* 相同目錄中。

5. 設置 *Other Audio* 下拉功能表爲 'Track 1' 。

6. 選擇 *Mono/Stereo* 單選按鈕。

7. 點擊 *Mono/Stereo* 文本區域旁的 *browse* 按鈕瀏覽和導入 *sampe_audio.wav* 文件。

8. 點擊 *Encode* 開始採用默認設置編碼實例。**注意：**可能會產生警告視頻持續時間長於音頻持續時間。忽略此警告。編碼過程仍然能順利進行。

9. 當編碼過程順利完成。*Preview* 按鈕有效。點擊該按鈕預覽 USM 編碼視頻。

10. 點擊視頻播放器左下方 *Audio Channel* 下拉式列示方塊方向按鈕，即在 *SubtitleAudio Channel* 下拉式列示方塊上方。此操作可以在聲道之間切換。

3.6 視頻設置

視頻設置單元用來設置其視頻質量等級、壓縮，使用 *Bitrate* 設置文件大小，以及設置幀速率和最終視頻的高度和寬度。

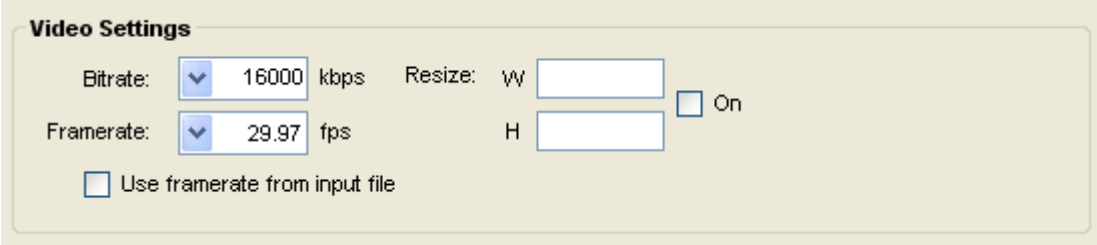

圖 7: 視頻設置

1. *Bitrate* 使用 — 或輸入期望視頻質量值。比特率爲一個每秒千比特的計算單位。決定了視頻壓縮的程度。比特率越高，視頻質量越好，文件尺寸也越大。比特率越低，壓縮率越高，視頻質量和文件尺寸也越低。

   以下列表表示了針對與不同平臺的編碼速率優化。通常最好使用低解析度和高比特率編碼應用，起先能夠加快編碼速度，然後在既定目標碼率和視頻版本準備好的情況下，創建一個高質解析度視頻文件。

不同平臺推薦比特率 (kbps)					
視頻格式		**PS3**	**Xbox360**	**Wii**	**PC**
1080p	高質量	40000	40000	n/a	40000
1080p	標準	30000	30000	n/a	30000
1080p	高壓縮	20000	20000	n/a	20000
720p	高質量	36000	36000	n/a	36000
720p	標準	26000	26000	n/a	26000
720p	高壓縮	16000	16000	n/a	16000
480p	高質量	16000	16000	8000	16000
480p	標準	12000	12000	6000	12000
480p	高壓縮	8000	8000	4000	8000

2. （可選）使用 *Framerate* 調整編碼視頻的幀速率。默認爲 29.97 fps— 標準 NTSC 視頻。當 AVI 導入到編碼器，*Use framerate from input file* 旁邊的標記選項被選中，編碼器將使用視頻原始幀速率。幀速率爲衡量視頻每秒鐘顯示的幀數量的值。標準 NTSC 電視信號幀速率爲 29.97 幀/秒，這是推薦的設置。PC 顯示器典型同步速率爲 60 幀/秒或者 60Hz。

3. （可選）使用 *Resize W* 和 *H* 文本域可以調整編碼視頻的輸出文件尺寸。輸入寬度 *W*，高度爲 *H*，置標記爲 *On*。

4. 點擊 *Encode* ，一旦編碼，點擊 *Preview* 播放視頻。

   編碼視頻將創建一個批量文件，位於輸出 USM 視頻文件相同的目錄。這個批量文件包含了使用編碼器命令行所必須的命令行。

5. 一旦編碼過程結束，點擊 *Preview* 播放視頻。

   可以通過禁止 *Encode* 按鈕旁邊的 *Preview in extended Video player option* 選項，使得預覽並受播放控制。否則使該選項有效。

4 Flash 中添加視頻入門

請注意：本章假定對 Flash 開發環境已經熟悉，且已經創建了 USM 編碼視頻文件。本章將一步步得介紹通過 ActionScript 腳本在 Flash 中添加視頻的過程。本章只作初步的介紹，對於添加視頻控制如播放、暫停、重播、視頻尺寸調整等詳細實現方法未作介紹。**關於這些操作的詳細資訊，請諮詢 Flash NetStream 類的幫助文檔。**

添加視頻的第一步爲用 Scaleform 編碼器編碼一個 USM 文件。請參考 Scaleform 視頻編碼入門獲得更多資訊。按照這裏的介紹步驟添加一個 USM 編碼視頻文件到 SWF 文件，並在 Scaleform Player 中播放 SWF 和視頻。

4.1 Flash 中視頻設置

創建一個空的 AS 2.0 Flash 文件，立即將文件保存在 Scaleform 視頻編碼器目錄。爲簡便，USM 視頻文件指導文件位於發佈 SWF 文件相同目錄。

1. 雙擊文字在時間軸上 *Layer 1* 的名字，修改爲 mvplayer，然後輸入新名字。

2. 使用 *New Layer* 按鈕在時間軸上創建一個新的圖層，取名爲 *'action'* 。

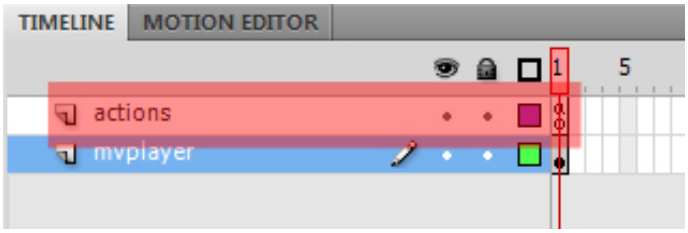

圖 8: 時間軸上的 mvplayer 圖層

3. 選擇時間軸上 *actions* 圖層的第一幀，按下（F9）打開 *Actions* 面板，在 *Actions* 中輸入以下代碼：

```
_global.gfxExtensions = true;
_focusrect = false;
onLoad = function()
{
      mvplayer.playVideo("sample.usm");
}
```

注釋：視頻文件存儲在發佈的 SWF 文件相同的目錄下，只要輸入視頻檔案名稱，包括 USM 副檔名。但是，如果視頻文件保存在其他目錄，需要輸入完整的目錄名，注意反斜杠的正確寫法 — 例如：*"C:/pathtovideo/sample.usm"* 。

4. 選擇時間軸上的 *mvplayer* 層。

5. 在場景上創建一個動畫剪輯。
   a. 實現的方法之一爲繪製一個大黑色邊線的框填充滿整個場景，透明填充。
   b. 在黑色邊框上雙擊選中整個框。
   c. 用滑鼠游標直接點擊邊框任何位置，選擇 *Convert to Symbol* 。
   d. 新建動畫剪輯命名爲 'mvplayer' 並點擊 OK。

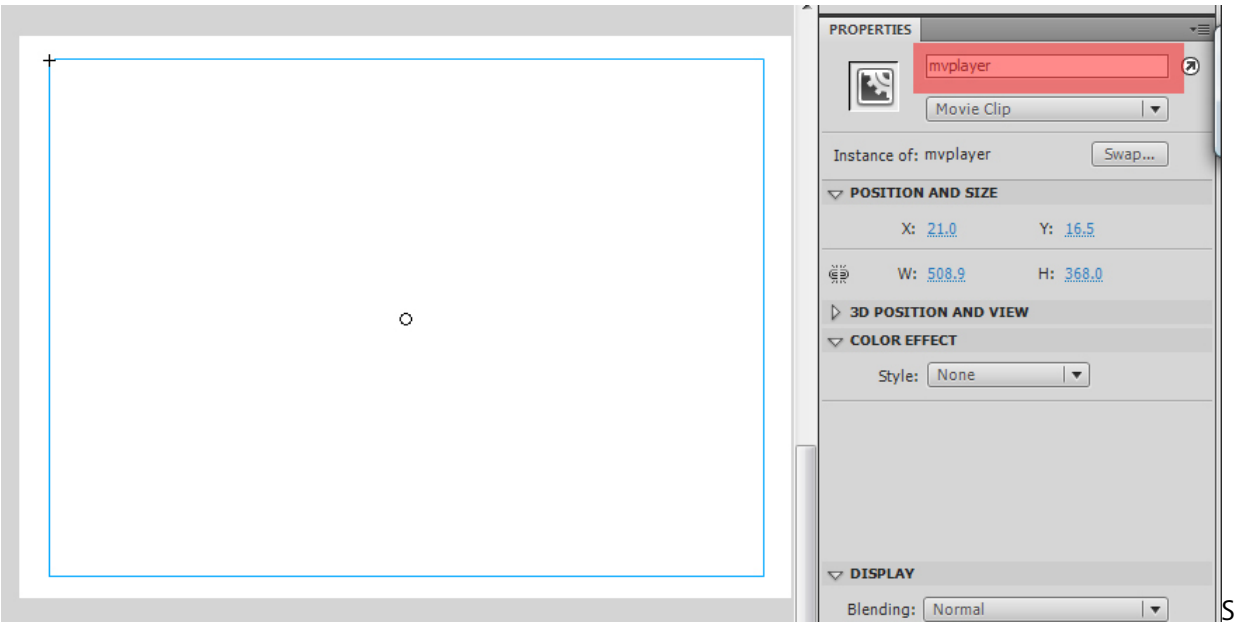

圖 9: 創建視頻剪輯 mvplayer。

6. 在場景上任何黑色邊框上單擊並選擇 new *mvplayer* 動畫剪輯。

7. 在 Properties 標簽下改變動畫剪輯實例名爲 'mvplayer' 。

8. 雙擊 *mvplayer* 動畫剪輯輸入時間軸。

9. 在 *mvplayer* 視頻剪輯內部，創建新的圖層，標簽名爲 *'actions'* 。

10. 選擇時間軸上 *actions 圖*層的幀 1，打開 *Actions* 面板，並輸入以下代碼：

```
nc = new NetConnection();
nc.connect(null);
ns = new NetStream(nc);
video.attachVideo(ns);
this.attachAudio(ns);
var audio_sound:Sound = new Sound(this);
function playVideo(videoToPlay:String):Void
{
      ns.play(videoToPlay);
}
```

注釋：確保 *mvplayer* 時間軸上 *actions* 圖層上的幀 1 被選中，在輸入以上代碼前， *Layer 1* 中不能有圖像。

11. 點擊 *Library* 面板，從彈出功能表中選擇 *New Video* 。

12. 新視頻命名爲 'Video' 並點擊 *OK*。

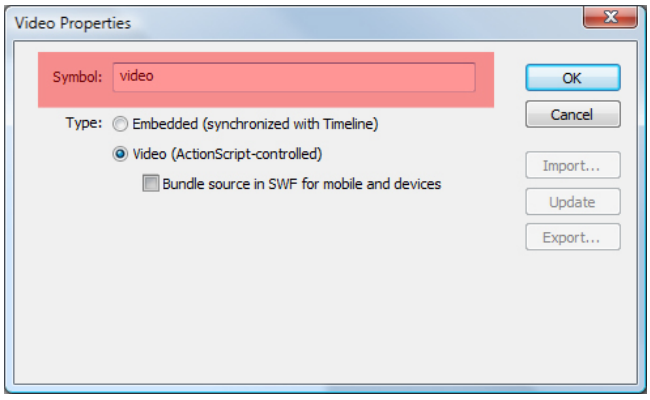

圖 10: 創建視頻物件

13. 使用 *New Layer* button 創建一個新圖層。本圖層最好命名爲 'Layer 3' 。

14. 從 *Library* 面板拖動一個新的視頻物件到 *Layer 3* 上場景的 *Library* 中。重新定位，並修改大小。

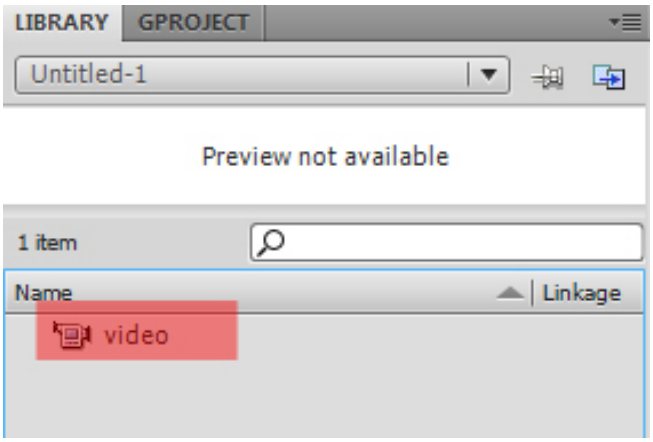

圖 11: Library 面板中的視頻物件。

15. 選擇場景中的視頻物件，在 *Properties* 標簽中改變實例名爲 'Video' 。

16. 保存並發佈動畫。

4.2 測試視頻

一旦創建了包含視頻的 SWF 文件，在 Scaleform Player 觀看視頻就非常容易了。

1. 通過 Flash 中的 Scaleform 啓動面板啓動 SWF。

或者：

1. 打開一個 Scaleform Player 窗口。

2. 將 SWF 文件拖動到 Scaleform Player 窗口。
    a. 也可以將 SWF 文件拖動到 Scaleform Player 的快捷鍵上。

5 ActionScript 視頻 應用入門

在創建了一個完整的 USM 編碼文件（包括了字幕、播放暗點和多音頻通道）後，視頻通過 ActionScript 添加到 Flash 文件，可以控制字幕的顯示，通過音頻通道改變聲音輸出，使用播放暗點進行導航和執行事件。注釋：本節基於指導文件 Flash 中添加視頻入門。請創建在例子中相關的 Flash 文件以按照指導手冊步驟來實施。

本節將通過實際使用 Flash SWF 中 Scaleform 視頻的播放暗點、字幕和音頻聲道來指導用戶其使用方法。本指導手冊意圖爲在 ActionScript 中實現視頻提供的基本方法介紹。有些示例代碼爲推薦部分，但不是實現的唯一方法。最後，需要具體的工程來決定代碼的執行。

5.1 ActionScript 中播放暗點的應用

USM 文件可用播放暗點編碼，使得視頻擁有導航和事件節點。導航節點可以用來對章節進行導航；事件節點用於發送鍵盤/值 參數給 Flash 文件，收到這些資訊可以觸發事件或者改變屬性。

注釋：一個 Flash SWF 擁有/播放 USM 視頻編碼，具有播放暗點文字檔案，需要能夠訪問播放暗點。如果按照文檔順序，應該已經創建了一個編碼實例包含播放暗點的 USM 視頻文件。如果沒有按照文檔順序操作，請參考添加視頻到 Flash 和 播放暗點添加播放暗點到視頻 相關文檔。

一些 ActionScript 的新代碼行必須能夠被播放暗點訪問。在 *mvplayer* 視頻剪輯（之前章節中已經創建）的 *actions* 圖層的幀 1 中輸入以下代碼。輸入或者拷貝代碼到動態腳本面板末尾。

1. 雙擊 *mvplayer* 動畫剪輯。

2. 在 *mvplayer* 中，選擇時間軸 *actions* 圖層的第一幀，在 *Actions* 面板中輸入代碼片段，放置在已有代碼下面。

3. 監聽播放暗點事件，監聽到事件，則執行函數 `traceCuePoint`。

```
ns.onCuePoint = traceCuePoint;
```

4. 函數 `traceCuePoint` 接收播放暗點物件爲一個參數。下一步，函數存儲該參數到 `metaPropPJParams` 物件。從那裏，函數跟蹤（顯示在輸出視窗）播放暗點基本資訊：播放暗點名 (`currentCuePoint.name`)、播放暗點時間 (`currentCuePoint.time`)和播放暗點類型 (`currentCuePoint.type`)。可以使用播放暗點資料控制視頻播放（聯合 `ns.time`），來代替簡單跟蹤執行代碼。

```
function traceCuePoint(currentCuePoint:Object):Void
{
      var metaPropPJParams:Object = currentCuePoint.parameters;
      trace("\t\t name: " + currentCuePoint.name);
      trace("\t\t time: " + currentCuePoint.time);
      trace("\t\t type: " + currentCuePoint.type);
      if (metaPropPJParams != undefined)
      {
            trace("\t\t parameters:");
            traceObject(metaPropPJParams, 4);
      }
}
```

5. 最後，如果播放暗點有參數，則通過調用下面的 `traceObject` 函數跟蹤那些參數到輸出視窗。播放暗點數據能夠用來改變 SWF 或觸發事件的屬性（變數）。

```
function traceObject(obj:Object, indent:Number):Void
{
	var indentString:String = "";
	for (var j:Number = 0; j < indent; j++)
	{
		indentString += "\t";
	}
	for (var i:String in obj)
	{
		if (typeof(obj[i]) == "object")
		{
			trace(indentString + " " + i + ": [Object]");
			traceObject(obj[i], indent + 1);
		}
		else
	{
			trace(indentString + " " + i + ": " + obj[i]);
		}
	}
}
```

6. 發佈和測試 Flash 文件，可以在 Scaleform Player 中通過 Scaleform 啓動面板或者拖動 SWF 文件到 Scaleform Player 窗口。如果每個步驟都正確執行，則視頻播放時視頻播放暗點應該可以在 Scaleform Player 輸出視窗被觀察到。

5.2 ActionScript 中字幕應用

字幕能夠在作爲視頻的 Flash 中操作和顯示，只需要視頻編碼的時添加了字幕，就像電影中的配音字幕一樣。在視頻中支援多語言字幕。

注釋：在使用字幕視頻文件之前，一個 Flash SWF 必須創建擁有/播放一個包含字幕文本的 USM 視頻編碼文件。如果你根據文檔描述步驟，應該已經創建了一個完整的包含兩種字幕的視頻編碼實例文件 *sample.usm*。如果沒按照步驟操作，請參考 添加視頻到 Flash 和 添加字幕到 Flash 相關章節。

1. 雙擊 *mvplayer* 動畫剪輯。

2. 在 *mvplayer* 中， 選擇時間軸上 *actions* 圖層第一幀，輸入代碼片段到 *Actions* 面板中，在已有代碼後面。

3. 首先，創建一個變數，`subtitleChannelNumber`，記錄字幕通道數量，設置爲 `0`：

```
var subtitleChannelNumber = 0;
```

4. 下一步，從視頻文件元資料中監聽字幕通道數目。注釋：字幕通道值爲 0 表示字幕通道關閉。

```
ns.onMetaData = function(infoObject:Object)
{
      ns.subtitleTrack = 1;
      subtitleChannelNumber = infoObject.subtitleTracksNumber;
      trace("Subtitle Channels: " +  subtitleChannelNumber);
}
```

5. 在場景中創建一個動態文本域，在 *mvplayer* 動畫剪輯內部，設定其實例名稱爲 *'subtitle'*。保證容量夠大，並有明顯的顏色和字體選擇。

6. 以下調用函數在內嵌字幕的視頻播放時觸發一個事件點時被執行。編碼後的 USM 文件中 Msg 字串包含了輸入的字幕文本內容。在第一行字幕，字串表示的內容爲輸入到字幕文件的內容。第二行字幕，爲變數名 `subtitleId2`，變數名用來決定通過 ActionScript 將要顯示的消息。Msg 中的內容將在第 5 步中創建的 *subtitle* dynamic 文本區域顯示。

```
ns.onSubtitle = function(msg:String)
{
      trace("Subtitle: " + msg);
      subtitle.text = msg;
 }
```

7. 發佈和測試 Flash 文件，可以在 Scaleform Player 中通過 Scaleform 啓動面板或者拖動 SWF 文件到 Scaleform Player 窗口。如果每個步驟都正確執行，則視頻字幕在視頻播放時在輸出視窗和 Flash 文件文本區域中都可以顯示出來。

5.3 在 ActionScript 音頻聲道應用

USM 可以爲多音頻聲道編碼，使視頻包含獨立的聲道應用於不同的語言。在 SWF 中訪問這些音頻聲道過程非常簡單，只需要很少的代碼。

訪問視頻文件的音頻聲道，一個 Flash SWF 必須創建一個擁有/播放一個至少包含單通道的 USM 視頻文件。該過程的簡要介紹，請參考 Flash 中視頻添加和視頻音頻通道編碼相關章節。

在 *mvplayer* 動畫剪輯中放置以下代碼到 *actions* 圖層的幀 1 中。輸入或者拷貝代碼到 *Actions* 圖層面板的末尾。

1. 如果尚爲準備好，雙擊 *mvplayer* 動畫剪輯輸入時間軸。

2. 在 *mvplayer* 內部，選擇時間軸 *actions* 圖層第一個關鍵幀，輸入代碼段到 *Actions* 面板，位於所有已有代碼之後。

3. 聲明兩個變數。第一個 `AudioTracks` 爲一個包含每個音頻通道的陣列。第二個， `CurAudioTrack`，將保存當前選擇通道，初始化這個變數爲 `0`。

```
var AudioTracks:Array;
var CurAudioTrack = 0;
```

4. 下一步，設置回收函數來監聽視頻元資料。分配視頻 `audioTracks` 元資料給 `audioTracks` 陣列。

   注釋：如果‘`ns.onMetaData`’在增加字幕控制小節已經存在，在代碼塊內部增加代碼行—‘`AudioTracks = infoObject.audioTracks;`’—代碼添加在代碼行‘`trace("Subtitle Channels: " + subtitleChannelNumber);`’後面，在結束大括弧‘}’前面。

```
ns.onMetaData = function(infoObject:Object)
{
      AudioTracks = infoObject.audioTracks;
}
```

5. 爲測試用，在 *mvplayer* 動畫剪輯中的場景裏創建一個簡單的正方形圖動畫剪輯，可以當作爲一個按鈕。設置動畫剪輯實例名稱爲 ‘prev_audiotrack’ 。

6. 在 ‘mvplayer’ 中 ‘action’ 圖層中的第一個關鍵幀中輸入以下代碼到 *Actions* 面板。這些代碼在 next_audiotrack 按鈕有效時能夠迴圈播放音頻通道。

```
prev_audiotrack.onRelease = function()
{
```

```
        if (AudioTracks  == undefined || AudioTracks.length < 2)
{
               return;
        }
        if (CurAudioTrack == 0)
        {
              CurAudioTrack = AudioTracks.length - 1;
        }
        else
        {
               CurAudioTrack--;
        }
        ns.audioTrack = AudioTracks[CurAudioTrack].trackIndex;
}
```

7. 在 *mvplayer* 的場景中從步驟 5 複製動畫剪輯，取其實例名稱爲 ‘next_audiotrack’ 。

8. 在 ‘mvplayer’ 中 ‘action’ 圖層中的第一個關鍵幀中輸入以下代碼到 *Actions* 面板。這些代碼在 next_audiotrack 按鈕有效時能夠迴圈播放音頻通道。

```
next_audiotrack.onRelease = function()
{
      if (AudioTracks  == undefined || AudioTracks.length < 2)
      {
            return;
      }
      CurAudioTrack++;
      if (CurAudioTrack >= AudioTracks.length)
      {
            CurAudioTrack = 0;
      }
      ns.audioTrack = AudioTracks[CurAudioTrack].trackIndex;
}
```

9. 在 Scaleform Player 中發佈並測試 Flash 文件。如果所有步驟執行正確，播放視頻的各種類型的音頻聲道可以預覽，只需點擊 *Next* 和 *Previous* 音頻按鈕（如果編碼的 USM 文件包含多聲道；只能播放單個聲道）。

6 ActionScript 視頻擴展

在支援 Scaleform 的 Flash 中有許多 ActionScript 方法用來播放視頻文件。另外，Scaleform 還有一些新的擴展專用功能。搜索關於 ‘NetStream’ 的 Adobe 幫助文件可以查看更多關於內嵌 NetStream 屬性、方法和事件的資訊。

6.1 支援內嵌 NetStream 屬性

`currentFps:Number`

每秒顯示的幀數。

示例： `var currentFps:Number = ns.currentFps;`

`time:Number`

.播放進度，以秒爲單位。

例子： `var currentTime:Number = ns.time;`

6.2 支援內嵌 NetStream 事件

`onCuePoint = function(infoObject:Object) {}`

播放 USM 文件時當到達內嵌播放暗點位置時調用。

注釋： 在 Adobe Flash Help 中搜索 ‘onCuePoint’ 獲得更多資訊。

`onMetaData = function(infoObject:Object) {}`

USM 文件播放時當播放器接收到內嵌描述資訊時調用。

注釋： 在 Adobe Flash Help 中搜索 ‘onMetaData’ 獲得更多資訊。

Scaleform Extensions

`audioTracks`

USM 文件中音頻聲道陣列。

例子：

```
ns.onMetaData = function(infoObject:Object)
{
      audioTrackArray = infoObject.audioTracks;
}
```

`subtitleTracksNumber`

USM 文件中字幕通道總數。

例子：

```
ns.onMetaData = function(infoObject:Object)
{
      numSubtitleChannels = infoObject.subtitleTracksNumber;
      trace("Subtitle Channels: " +  numSubtitleChannels);
}
```

```
onStatus = function(infoObject:Object) {}
```

每次 NetStream 物件的狀態改變或者錯誤消息提示時調用。

注釋： 在 Adobe Flash Help 中搜索‘onStatus’獲得更多資訊。

6.3 支援內嵌 NetStream 方法

```
close()
```

停止播放所有流資料，設置 ns.time 屬性爲 0，使流資料可被另外部分使用。

例子： `ns.close();`

```
pause([flag:Boolean])
```

暫停或者繼續流媒體播放。

例子： `ns.pause(true);`

```
play(name:Object, start:Number, len:Number, reset:Object)
```

開始播放外部視頻（USM）文件。

例子： `ns.play("myVideo.usm");`

`seek(offset:Number)`

搜索離流媒體開始處幾秒內最近的的關鍵。

例子： `ns.seek(50);`

`setBufferTime(bufferTime:Number)`

以秒為單位設置將要緩衝的電影資料的數量。GFx::Video::Video 從磁片緩衝足夠的資料,以實現流暢播放並減少磁片讀取次數。緩衝區大小以視頻位率 (Bitrate) 以及其他視頻參數為依據。預設情況下,此緩衝區大小將足以容納 1 秒播放。

示例： `ns.setBufferTime(2.5);`

6.4 新 Scaleform 屬性

`subtitleTrack = Number`

NetStream 中當前使用的字幕通道數。

例子： `ns.subtitleTrack = 1;`

`audioTrack = Number`

NetStream 使用的主音訊軌道。當視頻檔有多個音訊軌道(例如,一個英語版本和一個西班牙語版本)時,主音訊軌道可用來針對特定語言設置音訊軌道。

示例： `ns.audioTrack = 2;`

`subAudioTrack = Number`

一個次要或 SubAudio 軌道,一般由 NetStream 用來隨電影一起播放對話軌道或音效。

示例： `ns.subAudioTrack = 3;`

`voiceTrack = Number`

取代 5.1ch 音訊的中心軌道。僅使用此來更改 5.1ch 音樂的語音軌道。

示例： `ns.voiceTrack = 1;`

`setReloadThresholdTime(reloadTime:Number)`

設置重新載入時間。確定從磁片重新填充視頻資料緩衝區的頻率。例如,如果一個應用程式將緩衝區大小設置為 4 秒(通過 setBufferTime),而把重新載入閾值 (Reload Threshold) 設置為 1 秒,則 GFx:;Video::Video 將會先用 4 秒長的資料填充緩衝區。在 3 秒長資料解碼並使用後,就剩下不到重新載入閾值秒長的資料,而 GFx::Video::Video 就會重新填充緩衝區。

示例： `ns.setReloadThresholdTime(1);`

`setNumberOfFramePools(numPools:Number)`

設置內部視訊緩衝區的數目。GFx::Video::Video 使用內部記憶體在顯示前緩衝已解碼的幀。較多幀池有助於在高 CPU 負載下理順播放活動。預設值為 2。

示例： `ns.setNumberOfFramePools(3);`

`setOpenTimeout(timeout:Number)`

以秒為單位設置打開檔超時(USM 頭解碼所必需的時間)。預設值為 5 秒。要禁用此功能(設置無限超時),請傳遞 0。

示例： `ns.setOpenTimeout(5);`

`currentFrame:Number`

返回當前播放位置(以幀計)。

示例： `trace("Current FPS: " + ns.currentFps + " fps");`

`Loop = Boolean`

設置迴圈爲 true 告知 NetStream 迴圈播放視頻，直到迴圈屬性設置爲 false。

例子： `ns.loop = true;`

`clipRect = new rectangle(x:Number, y:Number, width:Number, height:Number)`

用來顯示視頻的一個給定區域。

例子： `ns.clipRect = new rectangle(100, 100, 240, 200);`

`onSubtitle = function(msg:String) {}`

當播放器接受到一個來自 USM 文件的字幕字串時調用。

例子： `ns.onSubtitle = function(msg:String) { trace(msg); }`

7 視頻創建理解

視頻編輯或者視頻創建可以爲一個非常有技巧的過程，特別是對於那些對於視頻編輯和轉換沒有太多經驗的人來說尤其如此。視頻編輯包含了很多選項和設置，都將對視頻播放質量和性能産生影響。對於視頻，典型的需要考慮解析度、縱橫比、比特率、幀速率和其他相關因素，使其能夠以最佳質量畫面質量流暢播放。每個硬體平臺，使用案例和遊戲引擎在視頻編碼和保證高性能方面使用的方法略有不同。這需要在編碼過程中做一些實驗和測試。

例如：播放單個全屏視頻動畫需求較爲簡單，因爲作爲唯一運行程式可以利用系統大多數資源。但是，當需要用到多媒體流動畫、導入動畫背景、使用 alpha 效果、在紋理上播放動畫或者執行其他高級功能，則需要做一些不同方法的實驗來確定正確的設置。

當創建一個完整的 1080p 視頻可能是在 Xbox 360 和 PS3™上創建高質量動畫最好的方法，創建視頻之前對一些因素必須進行評估。首先，編輯 HD 視頻，因爲文件尺寸非常大，就會極其複雜且速度也會很慢。HD 視頻，特別是高比率率 1080p 視頻看起來效果非常好，但是文件很大，播放時佔據很多系統資源，所以必須將每個視頻中需要使用的解析度確定下來。

是否需要創建一個 1080p 的視頻動畫，其他需要考慮的因素爲：視頻原始檔案爲什麼（即時動畫、CG、遊戲內部畫面等）、視頻創作人、預算和開發周期。

非常重要的是檢查在 720p、1080p 和其他 HD 解析度中即時畫面質量的區別，通常情況下低解析度畫面看起來已經能滿足需求，且創建容易、文件較小（不會佔據很多硬碟空間）、容易編輯、動畫流更加高效且播放佔據更少記憶體空間。在創作過程中評估和測試非常重要，根據實際要求確定工程目標，不單是考慮視頻的播放情況，還要考慮視頻流和創建過程是否高效。

最好是設定視頻爲最終理想的解析度或者縮小規模。Scaleform 視頻編碼器能夠調整視頻大小，但是可能會損失質量、畫面拉伸或者其他因素。這在對低解析度視頻進行擴展時特別明顯；但在對高解析度視頻進行縮小播放時影響很小。

幾乎任何的視頻編輯軟體都可以與 Scaleform 視頻配合使用。爲達到更好效果，視頻必須爲未經壓縮的 AVI 文件。使用 Scaleform 視頻文件壓縮時需要使用未經壓縮的文件。如果 AVI 文件由另外的編碼器創建（如 DIVX）並且進行了視頻壓縮（進行藝術加工），這樣，將視頻文件轉換成 Scaleform 視頻格式時相當於進行了兩次壓縮，這樣不能減少視頻文件大小，反而會影響視頻質量。

創作視頻時，音頻也是一個重要因素。需要知道是否需要支援單聲道、身歷聲或者 5.1 聲道；但是，音頻定位可能更爲重要。需要考慮配音文件如何創建，特別是當需要爲每種不同語言翻譯和錄音，準備一個配音文件（不單是創建字幕）。Scaleform 視頻能夠採用多聲道音頻軌道，可以用來爲不同語言播放不

同配音，這樣可以在單個視頻文件中支援多種語言，而無需創建多個視頻文件包含單獨的語言。但是，使用這種方法，配音內容需要從視頻/音頻編輯程式中導出爲單獨的文件，這樣才可以在運行中進行交叉調用。

再次強調，在初始化視頻作品前，最好在整個開發過程中考慮到所有的遊戲視頻需求和性能的測試指標，以避免時間浪費或者長遠的性能問題。

7.1 Alpha 通道

對於視頻編輯和電腦圖像不熟悉的用戶可能對 alpha 通道或者蒙版概念不是很熟悉。Scaleform 視頻支援圖元點 alpha 通道，意思爲視頻中的每個圖元點可以爲實體顏色或者是一定程度的透明化顯示。在一個典型的視頻編輯程式中，視頻可以爲完全或者部分透明，這根據分配在通道上的控制透明程度的顏色而定。

這在通常的 TV 和電影裏使用的“綠光屏”中經常用到，其中演員以綠色實體（或者藍色）爲背景，這樣可以易於刪除背景，將演員合成到其他場景中。亮綠色或者藍色用的最多，因爲這兩種顏色最顯眼，且在電影的其他畫面中很少用到，所以不容易與銀幕其他畫面混淆。視頻畫面能夠以全透明或者部分透明編碼。

在大多數程式中增加一個 alpha 通道相對容易，增加 alpha 效果到視頻編碼猶如點擊一個檢驗按鈕那麼容易。但是，需要注意的是具有 alpha 效果的視頻在渲染上比沒有 alpha 效果要複雜兩倍。在遊戲中需謹慎使用 alpha 透明效果，特別是在即時遊戲引擎中。

8 技術集成指南

本章提供了集成 Scaleform 視頻到你所用的引擎需要瞭解的技術要點。

8.1 Scaleform 聲音系統初始化

初始化 Scaleform 聲音系統，需要將一個 GFx::Audio 類實例設置到 GFx::Loader 物件上。該狀態的目的爲提供聲音渲染物件，SoundRenderer 和播放 SWF 聲音流同步參數，SoundRenderer 爲一個抽象的 C++ 介面，應該在遊戲中用來產生聲音效果。Scaleform 默認爲需要執行，該元件基於 FMOD 跨平臺聲音庫並可以在所有支援的平臺上使用，要創建一個該實例，需要調用 Sound::SoundRendererFMOD::CreateSoundRenderer 方法，然後創建的物件應該用 FMOD::System 物件初始化。

例子：

```
// 初始化聲音渲染物件
FMOD::System* pFMOD = NULL;
result = FMOD::System_Create(&pFMOD);
if (result != FMOD_OK)
      exit(1);
result = pFMOD->getVersion(&version);
if (result != FMOD_OK || version < FMOD_VERSION)
      exit(1);
result = pFMOD->init(64, FMOD_INIT_NORMAL, 0);
if (result != FMOD_OK)
      exit(1);
GPtr<GSoundRenderer> psoundRenderer =
                        *GSoundRendererFMOD::CreateSoundRenderer();
if (!psoundRenderer->Initialize(pFMOD))
      exit(1);

// 在導入 GFx::Audio 上設置一個實例
Ptr<GFx::Audio> paudioState = *new GFx::Audio(psoundRenderer);
loader.SetAudio(paudioState);
```

8.2 視頻播放系統初始化

需要初始化 Scaleform 視頻播放系統，Video 類的一個實例應該設置到 GFx::Loader 物件。本狀態的目的爲當應用程式需要播放視頻文件時創建一個視頻播放器的實例。如果視頻文件包含音頻文件並需要播放，則 VideoSoundSystem 介面的一個實例需要被設置到 GFx::Video::Video 物件。Scaleform 爲每個支援的平臺提供了可執行的介面，就是以執行爲基礎的 SoundRenderer 介面。

例子：

```
// 導入中設置一個 Video 實例
Ptr<GFx::Video::Video> pvc = *new Video(Thread::NormalPriority);
// 設置基於 SoundRenderer 介面的視頻音頻系統實例
if (psoundRenderer)
{
      pvc->SetSoundSystem(psoundRenderer);
}
// 設置一個應用於特殊平臺的視頻音頻系統
// 注意：SetSoundSystem 方法應該在每個 Video 中調用一次。
pvc->SetSoundSystem(Ptr<Video::VideoSoundSystem>(*new
                                    VideoSoundSystemDX8(0)));

loader.SetVideo(pvc);
```

在Windows平臺,Video::Video類叫VideoPC然後需要三個參數。第一個參數是視頻解碼線的優先。第二個參數控制用多少個線來做視頻解碼,然後第三個參數是每一個解碼線的affinity mask。

```
VideoPC(Thread::ThreadPriority
            decodingThreadsPriority = Thread::NormalPriority,
            int decodingThreadsNumber = MAX_VIDEO_DECODING_THREADS,
            UInt32* affinityMask = NULL);
```

例子：

```
UInt32 affinities[] = {
                        DEFAULT_VIDEO_DECODING_AFFINITY_MASK,
                        DEFAULT_VIDEO_DECODING_AFFINITY_MASK,
                        DEFAULT_VIDEO_DECODING_AFFINITY_MASK };

Ptr<Video::Video> pvc = *new Video::VideoPC(Thread::NormalPriority,
                                    MAX_VIDEO_DECODING_THREADS, affinities);
```

對於 PS3 和 Xbox360 平臺，Scaleform 提供了特殊的 Video 類，包含一些額外的初始化參數。

對於 PS3，本狀態命名爲 GFx::Video::VideoPS3 並包含 5 個參數：

```
VideoPS3(CellSpurs* spurs,
             int ppu_num, Thread::ThreadPriority ppu_priority,
             int spu_num, UInt8* spurs_priorities = NULL);
```

spurs - CellSpurs 結構指標
spu_num - 用於視頻編碼的 SPU 數量
ppu_num - 用於視頻編碼的 PPU 線程數量
ppu_priority - PPU 線程優先順序
spurs_priorities - SPU 核的優先

例子：

```
UInt8 spurs_priorities[] = {1,1,1,1,0,0,0,0};
 Ptr<Video::Video> pvc = *new VideoPS3(GetSpurs(), 0, 1000,
                                                    4, spurs_priorities);
```

最低 PPU 佔用率示例：

可以初始化 Scaleform Video 來實現其 PPU 佔用率最小化。例如,可以用 50-60 FPS 播放 720p 視頻,PPU 佔用率不到 5%。當主執行緒忙到足以達到搶佔視頻解碼執行緒的程度時就會發生這種情況。例如,假如某個遊戲正在運行,主執行緒就會忙到足以造成這種情況的程度。

請牢記,PPU 對解碼執行緒的佔用情況取決於是否載入其他執行緒。如果主執行緒不忙,解碼執行緒就只佔用大約 40% 的 PPU。但在以下設置的情況下,假如主執行緒忙,它就會優先于解碼執行緒,而解碼執行緒就只使用 4-5% 的資源。事實上,假如優先順序設置正確,解碼執行緒的佔用率可低至 1%。

```
UInt8 spurs_priors[] = {0,0,1,1,1,1,0,0};
Ptr<Video::Video> pVideo = *new VideoPS3(GetSpurs(), 0,
                            Thread::BelowNormalPriority, 4, spurs_priors);
```

對於 Xbox360 而言,該狀態被命名為 VideoXbox360,並使用三種參數來規定使用哪種硬體執行緒進行視頻解碼。第一個參數規定了兩個執行緒,它們主要完成大部分內部工作,因此這些執行緒不應重疊。第二個參數控制解碼視頻線的優先。第三個參數控制視頻渲染器的硬體線。

```
VideoXbox360(unsigned decodingProcs = Xbox360_Proc1 | Xbox360_Proc3 |
                           Xbox360_Proc5, Thread::ThreadPriority
                           decodingThreadsPriority = Thread::NormalPriority,
                           Xbox360Proc readerProc = Xbox360_Proc2);
```

例子：

```
Ptr<Video::Video> pvc = *new VideoXbox360(Xbox360_Proc1 |
                                          Xbox360_Proc3|Xbox360_Proc5,
                                          Thread::NormalPriority,
                                          Xbox360_Proc2);
```

8.3 背景遊戲資料載入 API

在視頻重播時，遊戲資料的背景載入將通過回調介面 Video::VideoBase::ReadCallback 和 – VideoBase::IsIORequired、 VideoBase::EnableIO 等方法來執行。

ReadCallback 是一個回調介面，它的作用是在要求視頻進行讀取操作和完成時對該程式進行通知。

例子：

```
// 視頻系統的簡單讀取回調運行
class VideoReadCallback : public VideoBase::ReadCallback
{
      public:
      VideoReadCallback() {}
      virtual ~VideoReadCallback() {}

      virtual void OnReadRequested()
      {
          // 視頻讀取開始
          VideoReadStart = Timer::GetTicks();
          // ...
      }
      virtual void OnReadCompleted()
      {
          // 視頻讀取完成
          GameReadStart = Timer::GetTicks();
          // ...
      }

      UInt64  VideoReadStart;
      UInt64  GameReadStart;
};

// 初始化視頻系統
Ptr<Video::Video> pvc = *new Video::VideoPC(
                Thread::NormalPriority, MAX_VIDEO_DECODING_THREADS, affnts);

// 創建並設置視頻讀取回調
Ptr<VideoReadCallback> pvideoReadCallback = *new VideoReadCallback;
pvc->SetReadCallback(pvideoReadCallback);
```

IsIORequired 確定了視頻系統是否需要運行任何視頻資料讀取操作。只要這一方法的返回值爲真，該程式就應當立即停止所有磁片輸入輸出操作，並通過調用 *EnableIO* 來啓用視頻讀取操作。

EnableIO 爲有效資料的背景載入啓用/禁用視頻讀取操作。

例子：

```
if(pvc->IsIORequired())
{
    pDataLoader->Enable(false);
    pvc->EnableIO(true);
    fprintf(stderr, "Video is loading\n");
}

// ...

if(!pvc->IsIORequired())
{
    pvc->EnableIO(false);
    pDataLoader->Enable(true);
    fprintf(stderr, "Game data is loading\n");
}
```

視頻背景載入功能將通過兩個演示（VideoBGLoadFlash 和 VideoBGLoadData）來說明，它們的完整源代碼可在 *Apps\Samples\Video* 目錄中獲取。

- VideoBGLoadFlash – 這一演示說明了當某一視頻播放時，在您的程式中載入 Flash 內容的簡單運行過程。
- VideoBGLoadData – 這一演示爲播放視頻時如何向程式中載入任意資料提供了範例。

8.4 Scaleform VideoSoundSystem 介面

GFx::Video::VideoSoundSystem 是一個抽象介面，爲 Video::Video playback 重播提供音頻支援；開發者通過對這一類作出自己的版本來替代音頻運行。在播放視頻之前，需要爲這一類創建一個實例，並安裝 Video::SetSoundSystem()。通常情況下，特定平臺的運行通常能夠避免運行這一介面。

音頻系統介面包含在 Video 分佈中：

- Video::VideoSoundSystemDX8 – Windows 中的 DirectSound
- Video::VideoSoundSystemXA2 – Windows 與 Xbox360 中的 XAudio2

- Video::VideoSoundSystemPS3 – MultiStream for PS3
- Video::VideoSoundSystemWii – Wii 系統音頻
- Video::VideoSoundSystemFMOD - 基於 FMOD 的音頻介面
- Video::VideoSoundSystemWwise – 基於 Wwise 的音頻介面

在當前版本的 Scaleform 中，由於嵌入了 Flash 音頻重播，視頻與音頻支援是解耦的，允許在使用視頻時，不必要求一般的音頻引擎。爲了完成這一工作，視頻將通過一個獨立的 Video::VideoSoundSystem 類獲得支援，它與 Scaleform 中的其餘 Sound::SoundRenderer 是分開的（詳情請參閱第 8.1 節內容）。這意味著未來獲得視頻支援，你只需運行 VideoSoundSystem 和 VideoSound 類，這比 SoundRenderer 要簡單很多。請注意，如果你已經運行了 SoundRenderer，你可用它對 Video 進行直接初始化，因爲它提供了一個功能集。在某些情況下，你還可以對二者進行混合運行（如果自定義視頻音頻類比一般音頻引擎提供了更好的流媒體支援，那麼這一做法是很有幫助的）。

例子：

```
#include "Video/Video_VideoSoundSystemXA2.h"
pvc->SetSoundSystem(Ptr<Video::VideoSoundSystem>(*new VideoSoundSystemXA2(0, 0)));
```

對於視頻音頻支援而言，需要運行 GFx::Video::VideoSoundSystem 和 GFx::Video::VideoSound 。通常而言，在視頻初始化過程中，僅安裝 VideoSoundSystem 的一個實例。VideoSoundSystem 提出了一個單一方法， Create，用於創建 VideoSound 物件，代表獨立的視頻音頻流。當有新的視頻被打開後，Scaleform 就會調用這一函數（可能會同時播放多個視頻）。當每一個 VideoSound 物件創建完成後， Scaleform 將會調用各種函數來說明開始和停止音頻輸出。實際的音頻資料將通過 VideoSound::PCMStream 輪詢後傳遞給指定音頻後獲得。輪詢功能通常由一個單獨的線程來完成，該線程由 VideoSoundSystem 維護，從而爲啟動的音頻服務。

請注意，基於音頻輸入插件的 VideoSoundSystemWwise 可作爲 Wwise SDK 的一部分來運行，因爲 v2009.2.1 build 3271. Audiokinetic 提供了完整的源代碼和 Visual Studio 方案/專案。這一插件放置的位置爲： *SDK\samples\Plugins\AkAudioInput*.。詳情請參考 Wwise 文檔。 GFx::Video::Video 分佈不包括 Wwise SDK 的任何部分，並且應當單獨安裝。

例子：

```
#include "Video/Video_VideoSoundSystemWwise.h"
Ptr<Video::VideoSoundSystem> wwise =
                  Ptr<Video::VideoSoundSystem>(*new VideoSoundSystemWwise());
pvc->SetSoundSystem(wwise);
wwise->Update();
```

更多詳情請參考 Scaleform Player - FxPlayerAppBase.cpp、 FxSoundFMOD.cpp、 FxSoundWwise.cpp 等源代碼，這些源代碼的位置是 *Apps\Samples\FxPlayer* 和 *Apps\Samples\Common* 目錄，並運行音頻系統介面 Video_VideoSoundSystem*.cpp ，其位置是 *Src\Video* 目錄。

8.5 多語言視頻

標準的多語言視頻每種語言有多個音訊軌道。GFx::Video::Video 採用 ActionScript 視頻擴展,使使用者可以有效地處理多語言電影資料。例如,我們來看一個同時包含英語和西班牙語身歷聲音訊的視頻。播放此視頻時,可以為每種語言選擇一個音訊軌道。要選擇音訊軌道,請使用 NetStream 提供的 audioTrack 擴展。
示例： `ns.audioTrack = 2;`

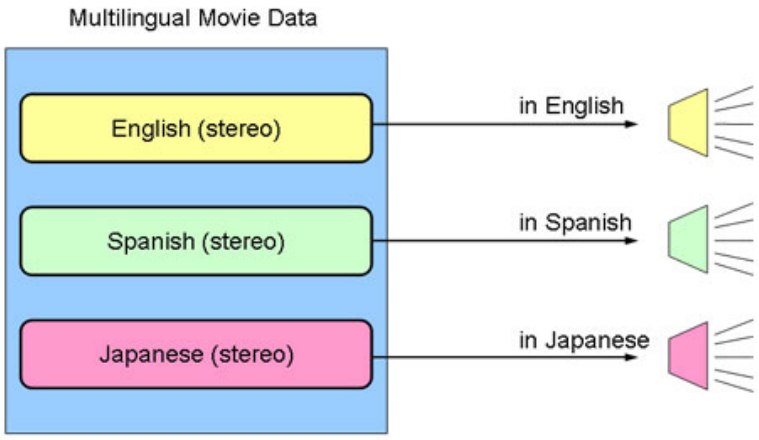

图 12：多语言视频

多語言環繞聲視頻是擁有一個通用 5.1 音訊軌道和多個語言軌道的視頻。儘管語言軌道可隨環繞聲一起切換,但只能將一個環繞聲軌道與多個語言軌道一起使用。這種在多個環繞視頻中共用單個聲音軌道的做法減小了電影大小。
GFx::Video::Video 支援以下兩種不同的播放功能：

1. 中心通道替換
2. SubAudio 播放

8.5.1 中心通道替換

如上所述,多語言環繞聲視頻減小了電影檔案大小以支援多種語言。多語言環繞視頻針對每種語言有一個通用的 5.1 音訊軌道和若干單聲道語音軌道。通常情況下,播放單聲道語音軌道,而不是 5.1 通道音訊的原始中心軌道。當播放多語言環繞聲視頻時,需要單獨選擇 5.1 音訊軌道和單聲道語音軌道。
如果要從中心通道播放音樂或聲音效果,必須提前將這些與語音軌道混合在一起。
示例：

```
ns.audioTrack = 0;     // main 5.1 audio track
ns.voiceTrack = 5;     // mono audio track
```

如果主軌道不是 5.1 音訊,或者語音軌道不是單聲道,就會忽略語音軌道的設置。如果將同一主音訊軌道設置為語音軌道,正常情況下會播放中心通道。

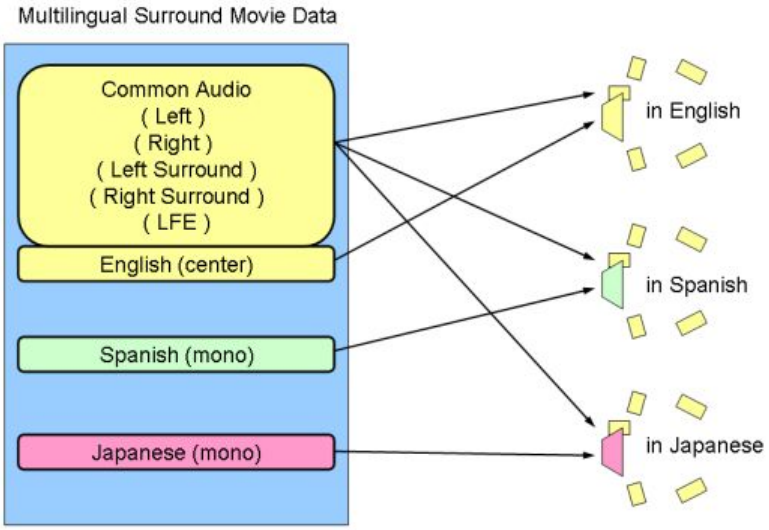

图 13：中心通道替换

8.5.2 SubAudio 播放

使用 SubAudio 播放功能時,應用程式可以通過 GFx::Video::Video 同時與主軌道一起播放另一個軌道。確切地說,GFx::Video::Video 通過一個附加聲音介面以及主軌道來輸出聲音資料。因此,應用程式可以通過作為語音軌道輸出 SubAudio 軌道來執行多語言環繞聲播放。

示例：

```
ns.audioTrack = 0;       // main audio track
ns.subAudioTrack = 6;    // secondary audio track
```

請關注: SubAudio Playback 能讓您在主音軌和副音軌裡獨立控制音效聲量。 只需要在裡用 Sound.setVolume() 擴展語法就能控制音效。

示例：

```
var audio:Sound = new Sound(this);
audio.setVolume(80, 50); // set volume for main and subaudio tracks
```

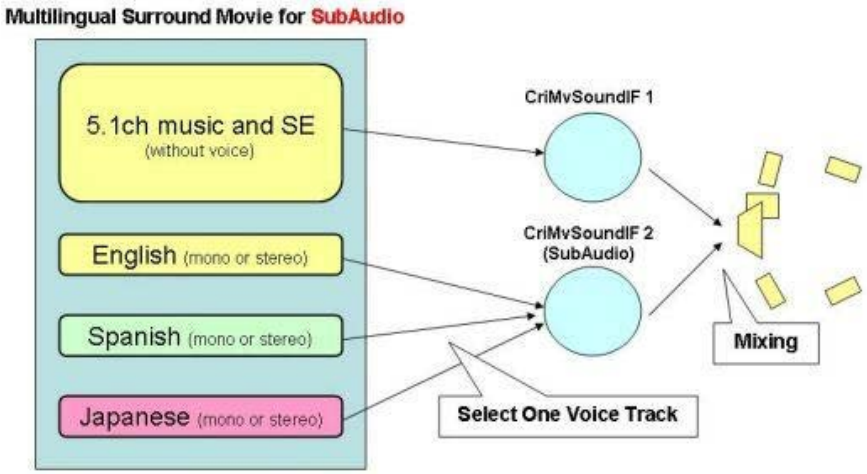

图 **14**：**SubAudio** 播放

9 故障排除

如果您的視頻內容播放不流暢,而且您正在遭遇時斷時續 (Stuttering),這通常是因為硬體沒有足夠可用資源來充分處理視頻資料。一般情況下,這可以通過執行以下操作進行修補:

1. 釋放您的代碼中的資源(即確保播放視頻時處理或 IO 負載不重),或者

2. 減小視頻資料的大小和位元速率,或者

3. 向視頻系統提供更多資源(執行緒、幀池)。

具體說來,下麵介紹三件事情(優先順序依次降低),通過調整這三件事情來說明確保視頻播放流暢：

1. 執行緒配置 – 更改數量和優先順序。看看您的遊戲引擎已經在用哪些執行緒,或者哪些尚未使用。較高解析度視頻(720P 以上)一般需要 3 到 4 個執行緒。有關配置執行緒的更多資訊,請參閱上面第 8 節。

2. 視頻解析度和位元速率 – 您可能需要減少您的視頻的大小和/或位元速率以減輕工作負載。Scaleform 將會根據需要自動按比例增加低解析度視頻以適應視窗。

3. 緩衝器和幀池 – 許多 ActionScript 方法和擴展可用於配置視頻播放。下麵的兩個調用可用來調節視頻系統使用的記憶體緩衝時間。有關更多詳細資訊,請參閱上面第 6 節。
   a. `NetStream.setNumberOfFramePools(numPools:Number)`
   b. `NetStream.setBufferTime(bufferTime:Number)`