Autodesk®
Scaleform®

Scaleform 视频入门

本文档详细描述了在 Scaleform 4.3 中 Scaleform 视频的使用

作者： Matthew Doyle, Vladislav Merker
版本： 3.02
上次修订：September 15, 2011

Autodesk®
GAMEWARE

Copyright Notice

Autodesk® Scaleform® 4.3

Autodesk Scaleform 联系方式：

文档	Scaleform 视频入门
地址	Scaleform Corporation 6305 Ivy Lane, Suite 310 Greenbelt, MD 20770, USA
网站	www.scaleform.com
邮箱	info@scaleform.com
电话	(301) 446-3200
传真	(301) 446-3199

目录

1 欢迎

Autodesk® Scaleform® Video™由 by CRI™提供，是一个集成到 Scaleform™的完整视频解决方案，是一个插件模块允许用户为 Adobe® Flash®增加高性能视频播放功能，充分利用了集成的 Flash 视频管道的优势。使用此新的视频模块，开发者可以播放高分辨率、高清晰度的画面，如标志、主菜单、HUD、游戏内绘制以及全屏影片场景切割。

CRI 视频编码器根据播放功能和相对于当前视频编码器的编码优势进行选择。CRI 视频的下一代播放引擎转为实时游戏系统构建，利用了最新多核硬件的优势。CRI 中间件已经在超过 1700 个游戏中获得应用。

Scaleform 视频允许视频在任何分辨率下播放，支持 PC 上和游戏控制器(Xbox 360®、PLAYSTATION®3"(PS3™)、Wii™)，全屏模式，窗体内，Alpha 透明效果以及更多其它功能。完全设计成流水线和多线程方式。使用 Scaleform 视频可以为不同的平台导出不同分辨率的视频以优化质量、帧速率、分辨率、比特率、尺寸、屏幕高宽比以及更多平台相关功能，确保视频质量和性能的优化。

Scaleform 视频工作流使得用户可以简单得在 Adobe Premiere®、Adobe After Effects®或者其他任何视频编辑应用工具中创建视频文件，可以将其导入到 Adobe Flash 并快速简单得导出到游戏中。在处理过程中，可以方便得创建标题、添加 5.1 声道音效和选择不同语言配音，交互小插件使游戏内视频交互功能更强。Scaleform 视频同样还有其他一些高级特性，如 alpha 通道、播放暗点，这些特性允许开发者利用视频功能不仅仅只是界面和场景分割。

Scaleform Video SDK 包括以下项：

- Scaleform 视频实时运行库
- Scaleform 视频工具
- 指南/实例
- Scaleform 视频入门(本文档)

2 关于 Scaleform 视频

Scaleform 视频是一种高分辨率、高声音质量视频播放系统，支持多平台应用。用来帮助创建符合全球最高标准，充分利用每个平台的特性的高质量视频。同时也具有高级特性，如多语种配音和字幕，游戏可以在不同的国家和地区销售。

2.1 安装位置

Scaleform 视频安装视频编码器位置为：

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

Scaleform 视频安装一个视频演示文件到以下位置：

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

本演示包括一个包含三个不同视频播放器的窗口，可以独立播放视频文件。本目录也包含一些演示文件相关的 Flash 文件和视频文件，包括一些预编码 USM 文件用来做测试用。在 Scaleform 播放器中运行 *videodemo.swf* 来试播演示文件。

2.2 指导文件

在编码器软件相同的位置，还有一些例子文件配合本指导文件使用。使用这些例子需要遵照指导文件中所描述的步骤。

- *getting_started_with_video_tutorial.fla* – 本 Flash 文件包含了本指导手册列出的所有内容。
- *getting_started_with_video_tutorial.swf* – 这是 Flash 文件的发布版本。
- *sample.avi* – 编码器的视频采样。
- *sample_audio.wav* – 编码器的音频采样。
- *sample_cue_points.txt* – 编码器的采样点。
- *sample_subtitles.txt* – 编码器的采样字幕。

2.3 编码器目录

在 Scaleform 编码器目录下有一些操作中非常关键的文件。

- Medianoche.exe — 编码器命令行。
- ScaleformVideoEncoder.exe — 编码器结束之前的图像(本文档中有所描述)。
- TMPGLib.dll — 编码器核心。
- VideoEncoderUtil.dll — ScaleformVideoEncoder.exe 必须要用的库。
- VideoPlayer.swf — 当在 Scaleform 视频编码器窗口中点击 *Preview* 时，这个即为图像视频播放器，控制视频文件输出。

2.4 优势

- **高质量视频播放**
  高质量视频能够使用简单的 API 函数进行播放。动画播放具有完全的高清晰度(HD)和分辨率以及良好的效果（例如 1080p）。

- **支持多语言配音和字幕**
  多语言配音和字幕可以放在单个视频文件中。为不同地区销售的游戏减少的开发量。

- **专业级编码简单易用**
  SDK 包中包含了一个简单易用的编码工具。

2.5 特性

Scaleform®视频，由 CRI Movie™提供，是一个插件模块，允许开发和通过 Adobe Flash®增加性能。使用 Scaleform 视频可以在多样化的下一代平台中播放极高质量、高分辨率和高清晰度视频。可以播放视频有：全屏、视窗、3D 游戏引擎中透明背景。在不同应用的交互内容中利用视频的主要包括：图标、主菜单、游戏 HUD、游戏纹理、游戏内视频反映、全屏电影场景切割、导入画面播放以及更多其它效果。在游戏内部播放视频不需要很多额外的工作，因为 Scaleform 视频已经完全集成进了 Scaleform™ 。
Scaleform 视频构建在当前非常流行并声誉卓著的 CRI Movie™编码器之上，且显著增强了其工作流并被 Adobe Flash 完全继承。

2.6 技术指标

- **视频和音频格式:** 编码器支持大多数常用的视频和音频格式，如 AVI 和 WAV。
- **多平台支持:** Scaleform 视频在多数主流平台上都支持包括 Xbox 360、PS3、PC 和 Wii，具有硬件独立性，可以自由编程。
- **不同平台的编码优化:** 在编码过程中，可以应用优化参数，这些参数将平台特性和视频质量/压缩率考虑在内。
- **字幕:** 同一个的影片支持多字幕通道。使得用户可以为不同的语言创建不同的字幕，存储在单独文件中，每种类型字幕都有不同的开始和结束时间进度。使用 Scaleform 字体配置系统多语言文本，能自动准确得定位。
- **Alpha 通道支持:** 用户可以在包含内嵌 alpha 允许按像素透明化以及整体透明化处理的动画中创建“虚拟透明”效果。这种方法可以使用户创造出多种不同且具有创意的效果。
- **纹理绘制:** 动画能够绘制在游戏纹理中以及 3D 图像表面，使得用户可以直接将视频集成到游戏当中。
- **音效声道:** 视频支持多音效声道（最多 32 个）。这使用户可以拥有多语言配音。
- **声道切换:** 用户可以使用 using Scaleform 4.3 Flash 扩展在 ActionScript™中不同声道之间切换。
- **定位支持:** 在 Scaleform 视频中通过 ActionScript 支持在运行时切换声道和字幕。这是一项新的特性，使用不同语言的客户可以更好的组织和管理 UI 视频。
- **环绕声支持:** 每个声道可以为单声道/立体声或者 5.1 立体环绕声。
- **多语言环绕声:** 环绕声中的中央通道数据可以被语音通道替代。这使用户可以简单得在环绕声中播放语音声道。
- **搜索能力:** 使用 Scaleform Video，用户能在视频流中搜索一个特殊的时间段内容。搜索定位值可以以视频流开始为参照或者当前位置为参照。用户能回退或者前进（使用正负搜索值）。
- **播放暗点:** 播放暗点为视频剪辑中重要的记号节点。播放暗点使得用户可以访问视频剪辑中不同的片段。使用 Scaleform 视频编码器，能够直接将播放暗点嵌入到 USM 文件。播放暗点也可以在 Flash 中使用。当到达视频的播放暗点时，将触发一个 ActionScript 事件，所有与该播放暗点相关联的信息传递给用户。这允许创建导航菜单、搜索视频中特殊事件以及使视频产生交互效果。

2.7 播放文件

Scaleform 视频将动画转换到为 Scaleform CRI 视频格式，文件后缀名为 USM。为了播放 USM 视频文件，只需将文件拖放到 Scaleform 媒体播放器图标即可，该图标应该在桌面上。也可以打开一个 Scaleform 播放器将 USM 文件拖放到打开的窗口中就能播放。

尝试拖放 *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

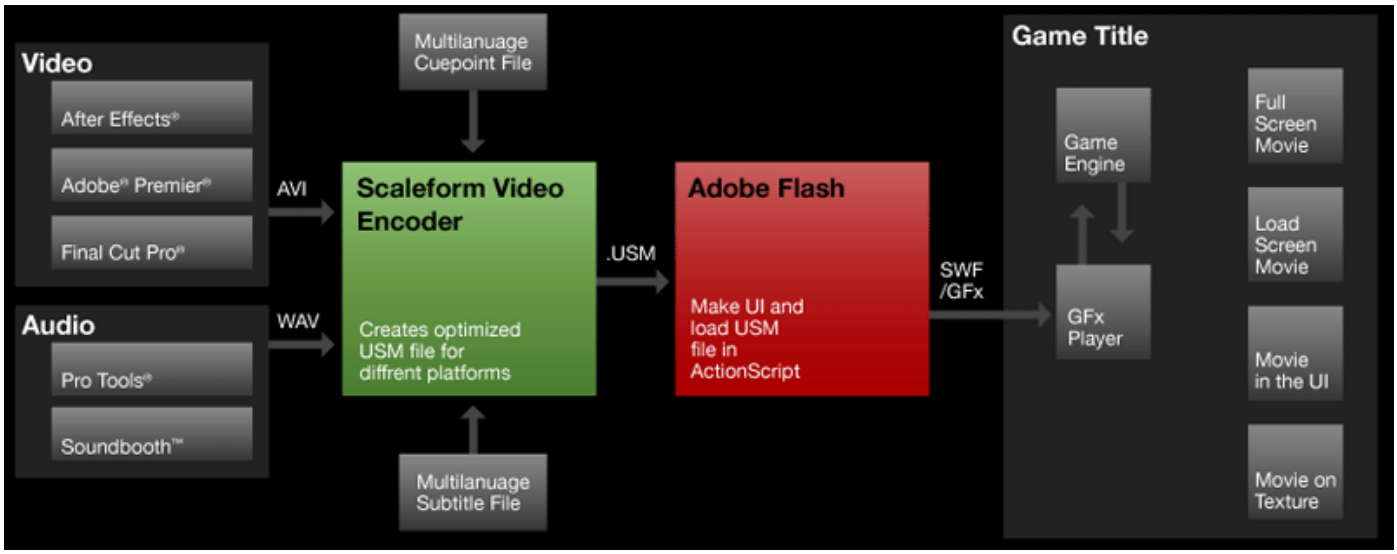

图 1：Scaleform 视频工作流

产生一个游戏内部的视频工作流如下所示：

1. 集成 Scaleform 视频 API 函数到游戏引擎.
2. 从 Adobe Premiere 或者其他视频编码器工具导出视频为 AVI 格式。
3. 使用 Scaleform 视频编码器转换 AVI 视频格式为 USM 格式。
4. 创建一个 Adobe Flash SWF 文件包含 USM 编码视频。
5. 创建一些 ActionScript 脚本来控制视频、字幕、播放暗点等。
6. 导入 SWF 文件到游戏。
7. 使用游戏引擎中可以的方法在游戏中播放 SWF 文件。

***注释：**不要试图将 USM 视频文件插入到 Flash 文件，可以通过 ActionScript 索引来代替。一旦创建了视频并在游戏中播放，视频中需要重复使用，无需在相同的位置和以相同的文件名重新对视频进行编码以更新任何 Flash 文件中内容。

3 Scaleform 视频编码入门

Scaleform™ 4.3 使用的视频必须编码成 CRI 的 USM 格式。CRI 格式为需要高质量画面和高分辨率下实现高性的游戏运行做了高度优化。为便于使用，Scaleform 提供了 Scaleform 视频编码器。本章将一步步解释如何使用编码器对 USM 文件进行编码，包括了对于播放暗点、字幕和声音集成的详细介绍。

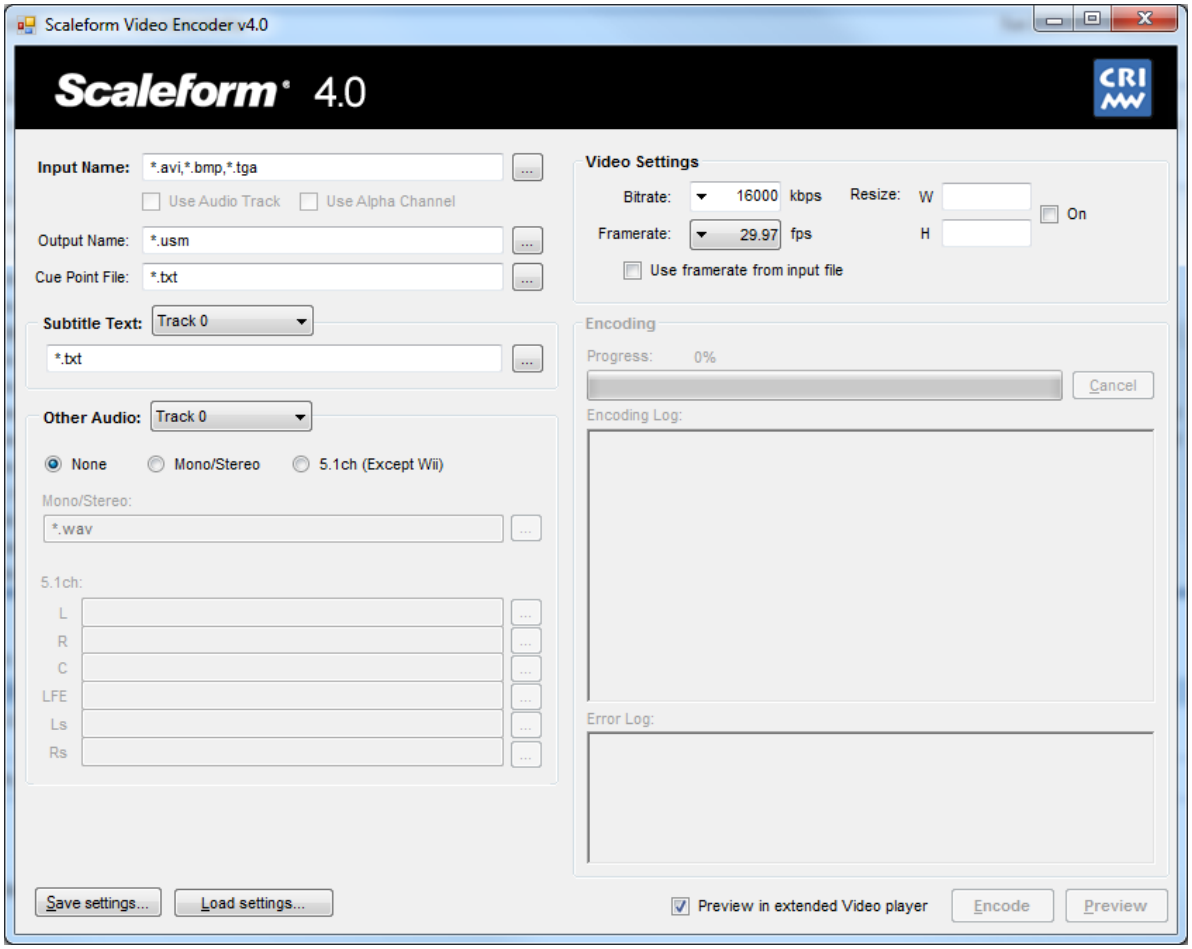

图 2: Scaleform 视频编码器

3.1 基本编码步骤

将视频转为 USM 格式过程非常简单。一些步骤，如字幕和播放暗点为可选步骤。接下来将描述创建 USM 视频文件所需的步骤，且包含了 alpha 通道（像素透明）、播放暗点、字幕和立体声（2 通道）功能。

使用 Windows™开始菜单启动 Scaleform 视频编码器。在 Scaleform 视频默认安装目录下，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* 按钮  来定位和导入 AVI 格式视频文件进行转换。
    a. 可能为 AVI 视频，或者 BMP 或 TGA 系列。
    b. 在 BMP 或 TGA 系列情况下，选择序列中的第一个文件。

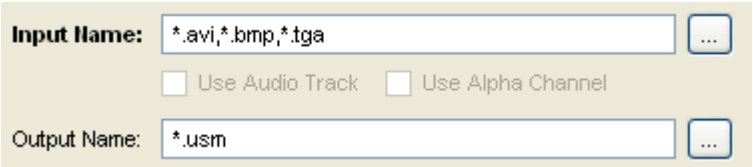

图 3: 视频文件编码必须输入名字

2. 如果视频有内嵌声道，则需要在 *Use Audio Track* 旁边放一个标记。这个选项为 AVI 默认提供的。在 BMP 和 TGA 序列中禁止使用。

3. 如果视频具有 alpha 透明通道则标记 *Use Alpha Channel* 项将其编码到最终 USM 视频文件。这样可以让观众看到视频之下 alpha 透明通道不是纯白色地方的内容。此选项模式情况下为未选中。

4. *Output Name* 域默认情况下为源视频路径；但最好是通过 *browse* 按钮浏览一个不同的目录保存 USM 编码文件。

5. 点击‘*Encode*’按钮启动编码。

3.2 第一个编码举例

1. 打开编码器。
2. 导入文件 *C:\Program Files\Scaleform\GFx SDK 4.3\Bin\Tools\VideoEncoder\sample.avi*。
3. 点击 *Encode* to 开始以默认设置对示例动画进行编码。
4. 当编码成功完成，*Preview* 按钮有效。按下该按钮预览 *USM* 编码视频。

3.3 播放暗点

播放暗点对于设置章节导航非常有用，用来在 SWF 文件中改变数据或者关闭事件。如果视频编码未包含播放暗点，可以跳到下一章节；但是，建议阅读播放暗点的简要介绍。

为了使最终的 USM 编码文件使用播放暗点，点击 *Cue Point File* 旁的 *Browse* 按钮定位一个播放暗点文本文件。

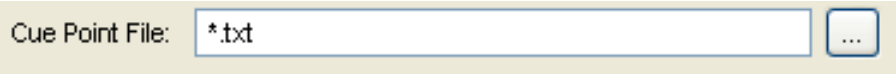

图 4: 浏览播放暗点文本文件。

一个播放暗点文本文件包括每个播放暗点排列成的行。序列第一行应包含毫秒间隔显示—典型为 10000。这个间隔用来作为决定每个播放暗点的时间值。一个时间值为 5000 即为 5 秒 — 5000/1000=5。间隔列表下的每行都有一个编码视频使用的单独播放暗点。

播放暗点有两种类型：导航和事件。导航播放暗点用来作为章节选择，很像 DVD 中的选段菜单。事件播放暗点用来设置参数。每个事件播放暗点可以有多个参数。参数列表为键=键值。单个播放暗点的多个参数由逗号隔开。关于如何在 Flash 文件中使用播放暗点的详细介绍，请参考 ActionScript 视频应用。

单行播放暗点格式

```
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 文件例子内容

```
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 用播放暗点重新对示例视频进行编码

1. 打开 Scaleform 视频编码器。

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

3. 点击 *Cue Point File* 文本区域的 *browse* 按钮，定位和导入 *sample_cue_points.txt* 文件 — 该文件位于 *sample.avi* 相同目录中。

4. 点击 *Encode* 用默认设置开始实例动画编码。

5. 当编码过程顺利结束，*Preview* 按钮有效。点击该按钮预览 USM 编码视频。

6. 在视频播放器中使用 *fast forward* 和 *rewind* 按钮进行播放暗点之间的快进和快退操作。

3.4 字幕

字幕文本文件包括了视频中的字幕行列表。字幕用来在视频的特定时间显示文本。如果视频不包含字幕，可以跳过本节到下一个章节；但是，建议阅读本节以理解如何添加字幕到视频中。

允许多种字幕文件 — 每个文件为一种字幕通道。这允许为每种语言创建一个字幕文件，有些语言根据语言中字幕长度和阅读所需时间需要不同的起始和结束时间。

1. 通过从 *Subtitle* 文本旁的下拉菜单中选择字幕轨道的使用。最多支持 32 轨道 —Tracks 0-31。

2. 点击 *browse* 选择字幕文本文件在轨道中使用。

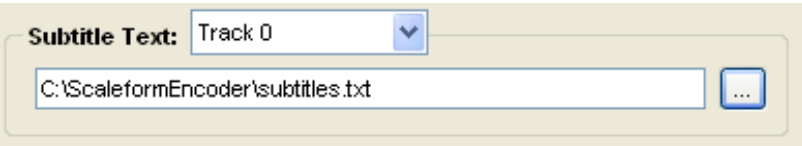

图 5: 选择字幕轨道并浏览文件。

字幕文件的第一行（不包括注释行）应该包含以毫秒为单位的显示间隔时间（典型为 1000）。这个值为每个字幕的开始和结束时间的基准。在第一行后面包括三部分信息：开始时间、结束时间、消息 ID。消息 ID 号在编码中用来显示文本字符串相关信息。同时，这个参数也可以被用来作为显示字幕的实际字符串。关于如何在 Flash 文件中使用字幕，请参考 ActionScript 视频应用。

单行字幕格式

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

Subtitle.txt 例子文件内容

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

3.4.1 视频例子添加字幕重新编码

1. 打开 Scaleform 视频编码器。

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

3. 点击 *Cue Point File* 文本区域的 *browse* 按钮，定位和导入 *sample_cue_points.txt* 文件 —该文件位于 *sample.avi* 相同目录中。

4. 点击 *Subtitle Text* 下拉菜单下面的文本区域旁的 browse 按钮。定位并导入 *sample_subtitles.txt* — 本文件位于 *sample.avi* 相同目录中。

5. 点击 *Encode* 用默认设置开始实例动画编码。

6. 当编码过程成功完成后，从您的桌面快捷方式启动 Scaleform Player（Scaleform媒体播放器），然后将 *getting_started_with_video_tutorial.swf* 文件拖入其中。

7. 注意字幕显示在视频播放器播放视频的底部。应该 2-5 秒产生一次，8-11 秒后再次产生。

3.5 音频

编码器的音频通道允许添加多种音频，包括单声道/立体声或者 5.1 环绕声。每个声道可以包括不同的音频文件，这可以用来进行多声道立体环绕声编码。

1. 首先，在 Scaleform 视频编码器 *Other Audio* 旁边的下拉菜单中选择一个音频轨道。

2. 选择音频类型 —无声、单声道/立体声或 5.1 声道（除 Wii 之外）。
   注意：如名字所述，5.1 声道在任天堂 Wii 中不可用；但是 5.1 声道在其他任何平台中都可以使用。

3. 对于单声道/立体声，使用 *Mono/Stereo* 文本区域旁边的 *Browse* 按钮，定位音频源文件为 WAV 格式。

4. 只要 *Use Audio Track* 有效，本区域已指定到了视频源文件所在路径。

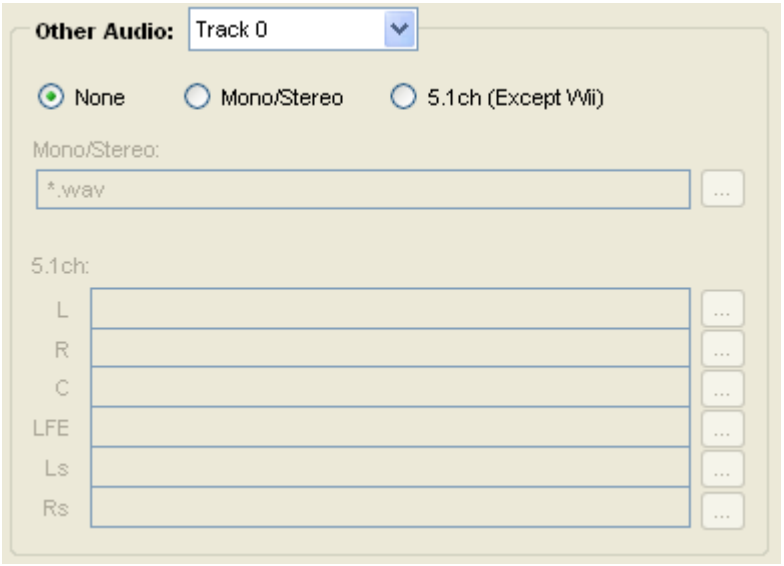

图 6: 选择字幕轨道并浏览查找一个文件

5. 对于 5.1 声道音频，使用每个通道列表旁的 *Browse* 按钮 — *L*、*R*、*C*、*LFE*、*Ls*、*Rs* — 为每个通道导入音频源文件。

使用 5.1 声道环绕声的通道：
L – 左声道
R – 右声道
C – 中央声道 (优先用于语音)
LFE – 重低音声道

Ls –后置左环绕声道
Rs – 后置右环绕声道

3.5.1 采用双声道音频轨道重新编码视频实例

1. 打开 Scaleform 视频编码器。

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

3. 点击 *Cue Point File* 文本区域的 *browse* 按钮，定位和导入 *sample_cue_points.txt* 文件 — 该文件位于 *sample.avi* 相同目录中。

4. 点击 *Subtitle Text* 下拉菜单下面的文本区域旁的 browse 按钮。定位并导入 *sample_subtitles.txt* — 本文件位于 *sample.avi* 相同目录中。

5. 设置 *Other Audio* 下拉菜单为 ‘Track 1’ 。

6. 选择 *Mono/Stereo* 单选按钮。

7. 点击 *Mono/Stereo* 文本区域旁的 *browse* 按钮浏览和导入 *sampe_audio.wav* 文件。

8. 点击 *Encode* 开始采用默认设置编码实例。**注意：**可能会产生警告视频持续时间长于音频持续时间。忽略此警告。编码过程仍然能顺利进行。

9. 当编码过程顺利完成。*Preview* 按钮有效。点击该按钮预览 USM 编码视频。

10. 点击视频播放器左下方 *Audio Channel* 组合框方向按钮，即在 *SubtitleAudio Channel* 组合框上方。此操作可以在声道之间切换。

3.6 视频设置

视频设置单元用来设置其视频质量等级、压缩，使用 *Bitrate* 设置文件大小，以及设置帧速率和最终视频的高度和宽度。

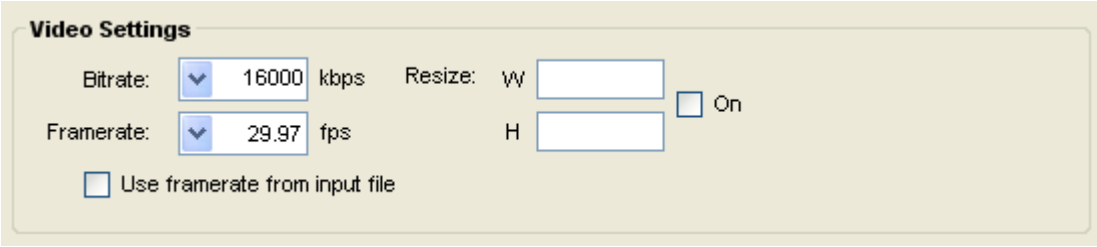

图 7: 视频设置

1. *Bitrate* 使用 — 或输入期望视频质量值。比特率为一个每秒千比特的计算单位。决定了视频压缩的程度。比特率越高，视频质量越好，文件尺寸也越大。比特率越低，压缩率越高，视频质量和文件尺寸也越低。

   以下列表表示了针对与不同平台的编码速率优化。通常最好使用低分辨率和高比特率编码应用，起先能够加快编码速度，然后在既定目标码率和视频版本准备好的情况下，创建一个高质分辨率视频文件。

不同平台推荐比特率 (kbps)					
视频格式		**PS3**	**Xbox360**	**Wii**	**PC**
1080p	高质量	40000	40000	n/a	40000
1080p	标准	30000	30000	n/a	30000
1080p	高压缩	20000	20000	n/a	20000
720p	高质量	36000	36000	n/a	36000
720p	标准	26000	26000	n/a	26000
720p	高压缩	16000	16000	n/a	16000
480p	高质量	16000	16000	8000	16000
480p	标准	12000	12000	6000	12000
480p	高压缩	8000	8000	4000	8000

2. （可选）使用 *Framerate* 调整编码视频的帧速率。默认为 29.97 fps— 标准 NTSC 视频。当 AVI 导入到编码器，*Use framerate from input file* 旁边的标记选项被选中，编码器将使用视频原始帧速

率。帧速率为衡量视频每秒钟显示的帧数量的值。标准 NTSC 电视信号帧速率为 29.97 帧/秒，这是推荐的设置。PC 显示器典型同步速率为 60 帧/秒或者 60Hz。

3. （可选）使用 *Resize W* 和 *H* 文本域可以调整编码视频的输出文件尺寸。输入宽度 *W*，高度为 *H*，置标记为 *On*。

4. 点击 *Encode* ，一旦编码，点击 *Preview* 播放视频。
   编码视频将创建一个批量文件，位于输出 USM 视频文件相同的目录。这个批量文件包含了使用编码器命令行所必须的命令行。

5. 一旦编码过程结束，点击 *Preview* 播放视频。
   可以通过禁止 *Encode* 按钮旁边的 *Preview in extended Video player option* 选项，使得预览并受播放控制。否则使该选项有效。

4 Flash 中添加视频入门

请注意：本章假定对 Flash 开发环境已经熟悉，且已经创建了 USM 编码视频文件。本章将一步步得介绍通过 ActionScript 脚本在 Flash 中添加视频的过程。本章只作初步的介绍，对于添加视频控制如播放、暂停、回放、视频尺寸调整等详细实现方法未作介绍。**关于这些操作的详细信息，请咨询 Flash NetStream 类的帮助文档。**

添加视频的第一步为用 Scaleform 编码器编码一个 USM 文件。请参考 Scaleform 视频编码入门获得更多信息。按照这里的介绍步骤添加一个 USM 编码视频文件到 SWF 文件，并在 Scaleform Player 中播放 SWF 和视频。

4.1 Flash 中视频设置

创建一个空的 AS 2.0 Flash 文件，立即将文件保存在 Scaleform 视频编码器目录。为简便，USM 视频文件指导文件位于发布 SWF 文件相同目录。

1. 双击文字在时间轴上 *Layer 1* 的名字，修改为 mvplayer，然后输入新名字。

2. 使用 *New Layer* 按钮在时间轴上创建一个新的图层，取名为 *‘action’*。

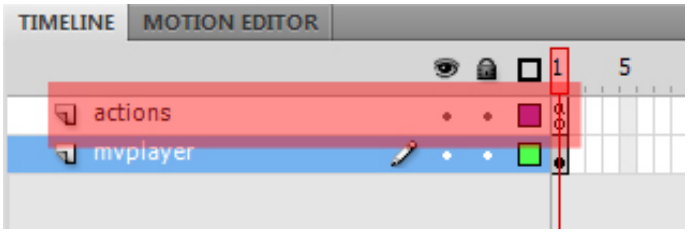

图 8: 时间轴上的 mvplayer 图层

3. 选择时间轴上 *actions* 图层的第一帧，按下（F9）打开 *Actions* 面板，在 *Actions* 中输入以下代码：

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

注释：视频文件存储在发布的 SWF 文件相同的目录下，只要输入视频文件名称，包括 USM 扩展名。但是，如果视频文件保存在其他目录，需要输入完整的目录名，注意反斜杠的正确写法 — 例如：*“C:/pathtovideo/sample.usm”*。

4. 选择时间轴上的 *mvplayer* 层。

5. 在场景上创建一个动画剪辑。
   a. 实现的方法之一为绘制一个大黑色边线的框填充满整个场景，透明填充。
   b. 在黑色边框上双击选中整个框。
   c. 用鼠标光标直接点击边框任何位置，选择 *Convert to Symbol*。
   d. 新建动画剪辑命名为‘mvplayer’并点击 OK。

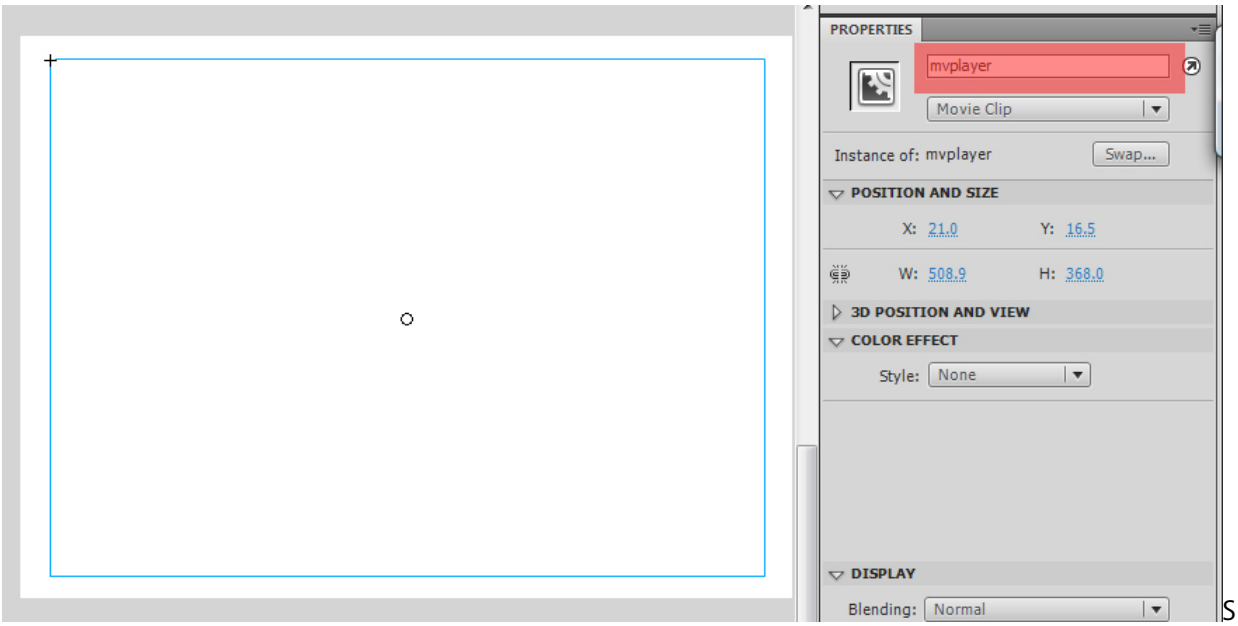

图 9: 创建视频剪辑 mvplayer。

6. 在场景上任何黑色边框上单击并选择 new *mvplayer* 动画剪辑。

7. 在 Properties 标签下改变动画剪辑实例名为‘mvplayer’。

8. 双击 *mvplayer* 动画剪辑输入时间轴。

9. 在 *mvplayer* 视频剪辑内部，创建新的图层，标签名为 *‘actions’*。

10. 选择时间轴上 *actions 图*层的帧 1，打开 *Actions* 面板，并输入以下代码：

```
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);
}
```

注释：确保 *mvplayer* 时间轴上 *actions* 图层上的帧 1 被选中，在输入以上代码前， *Layer 1* 中不能有图像。

11. 点击 *Library* 面板，从弹出菜单中选择 *New Video*。

12. 新视频命名为‘Video’并点击 *OK*。

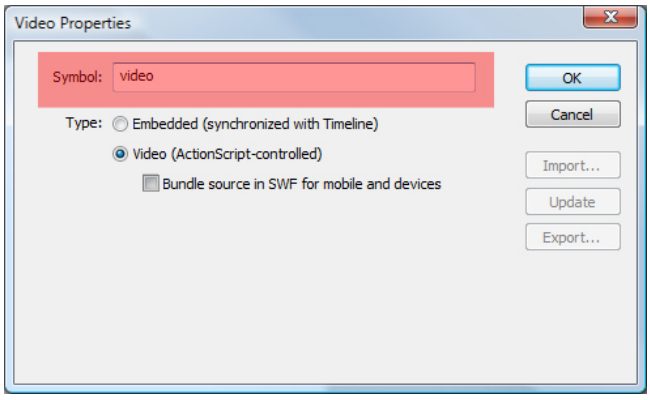

图 10: 创建视频对象

13. 使用 *New Layer* button 创建一个新图层。本图层最好命名为‘Layer 3’。

14. 从 *Library* 面板拖动一个新的视频对象到 *Layer 3* 上场景的 *Library* 中。重新定位，并修改大小。

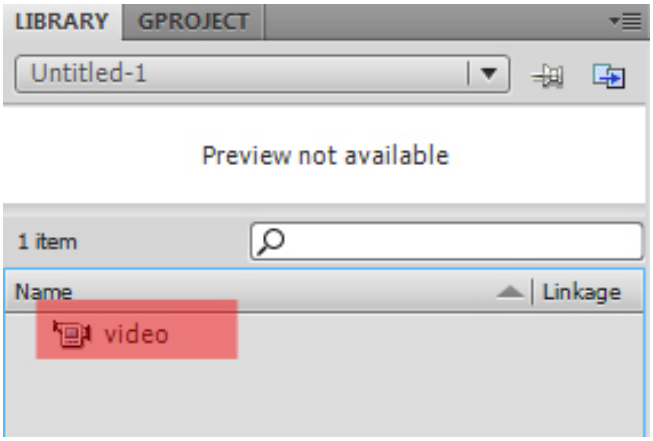

图 11: Library 面板中的视频对象。

15. 选择场景中的视频对象，在 *Properties* 标签中改变实例名为 *‘Video’*。

16. 保存并发布动画。

4.2 测试视频

一旦创建了包含视频的 SWF 文件，在 Scaleform Player 观看视频就非常容易了。

1. 通过 Flash 中的 Scaleform 启动面板启动 SWF。

或者：

1. 打开一个 Scaleform Player 窗口。

2. 将 SWF 文件拖动到 Scaleform Player 窗口。
    a. 也可以将 SWF 文件拖动到 Scaleform Player 的快捷键上。

5 ActionScript 视频 应用入门

在创建了一个完整的 USM 编码文件（包括了字幕、播放暗点和多音频通道）后，视频通过 ActionScript 添加到 Flash 文件，可以控制字幕的显示，通过音频通道改变声音输出，使用播放暗点进行导航和执行事件。注释：本节基于指导文件 Flash 中添加视频入门。请创建在例子中相关的 Flash 文件以按照指导手册步骤来实施。

本节将通过实际使用 Flash SWF 中 Scaleform 视频的播放暗点、字幕和音频声道来指导用户其使用方法。本指导手册意图为在 ActionScript 中实现视频提供的基本方法介绍。有些示例代码为推荐部分，但不是实现的唯一方法。最后，需要具体的工程来决定代码的执行。

5.1 ActionScript 中播放暗点的应用

USM 文件可用播放暗点编码，使得视频拥有导航和事件节点。导航节点可以用来对章节进行导航；事件节点用于发送键盘/值 参数给 Flash 文件，收到这些信息可以触发事件或者改变属性。

注释：一个 Flash SWF 拥有/播放 USM 视频编码，具有播放暗点文本文件，需要能够访问播放暗点。如果按照文档顺序，应该已经创建了一个编码实例包含播放暗点的 USM 视频文件。如果没有按照文档顺序操作，请参考添加视频到 Flash 和 播放暗点添加播放暗点到视频 相关文档。

一些 ActionScript 的新代码行必须能够被播放暗点访问。在 *mvplayer* 视频剪辑（之前章节中已经创建）的 *actions* 图层的帧 1 中输入以下代码。输入或者拷贝代码到动态脚本面板末尾。

1. 双击 *mvplayer* 动画剪辑。

2. 在 *mvplayer* 中，选择时间轴 *actions* 图层的第一帧，在 *Actions* 面板中输入代码片段，放置在已有代码下面。

3. 监听播放暗点事件，监听到事件，则执行函数 `traceCuePoint`。

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

4. 函数 `traceCuePoint` 接收播放暗点对象为一个参数。下一步，函数存储该参数到 `metaPropPJParams` 对象。从那里，函数跟踪（显示在输出窗口）播放暗点基本信息：播放暗点名 (`currentCuePoint.name`)、播放暗点时间 (`currentCuePoint.time`)和播放暗点类型 (`currentCuePoint.type`)。可以使用播放暗点数据控制视频播放（联合 `ns.time`），来代替简单跟踪执行代码。

```
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. 最后，如果播放暗点有参数，则通过调用下面的 `traceObject` 函数跟踪那些参数到输出窗口。播放暗点数据能够用来改变 SWF 或触发事件的属性（变量）。

```
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. 发布和测试 Flash 文件，可以在 Scaleform Player 中通过 Scaleform 启动面板或者拖动 SWF 文件到 Scaleform Player 窗口。如果每个步骤都正确执行，则视频播放时视频播放暗点应该可以在 Scaleform Player 输出窗口被观察到。

5.2 ActionScript 中字幕应用

字幕能够在作为视频的 Flash 中操作和显示，只需要视频编码的时添加了字幕，就像电影中的配音字幕一样。在视频中支持多语言字幕。

注释：在使用字幕视频文件之前，一个 Flash SWF 必须创建拥有/播放一个包含字幕文本的 USM 视频编码文件。如果你根据文档描述步骤，应该已经创建了一个完整的包含两种字幕的视频编码实例文件 *sample.usm*。如果没按照步骤操作，请参考 添加视频到 Flash 和 添加字幕到 Flash 相关章节。

1. 双击 *mvplayer* 动画剪辑。

2. 在 *mvplayer* 中， 选择时间轴上 *actions* 图层第一帧，输入代码片段到 *Actions* 面板中，在已有代码后面。

3. 首先，创建一个变量，`subtitleChannelNumber`，记录字幕通道数量，设置为 `0`：

```
var subtitleChannelNumber = 0;
```

4. 下一步，从视频文件元数据中监听字幕通道数目。注释：字幕通道值为 0 表示字幕通道关闭。

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

5. 在场景中创建一个动态文本域，在 *mvplayer* 动画剪辑内部，设定其实例名称为‘*subtitle*’。保证容量够大，并有明显的颜色和字体选择。

6. 以下调用函数在内嵌字幕的视频播放时触发一个事件点时被执行。编码后的 USM 文件中 Msg 字符串包含了输入的字幕文本内容。在第一行字幕，字符串表示的内容为输入到字幕文件的内容。第二行字幕，为变量名 `subtitleId2`，变量名用来决定通过 ActionScript 将要显示的消息。Msg 中的内容将在第 5 步中创建的 *subtitle* dynamic 文本区域显示。

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

7. 发布和测试 Flash 文件，可以在 Scaleform Player 中通过 Scaleform 启动面板或者拖动 SWF 文件到 Scaleform Player 窗口。如果每个步骤都正确执行，则视频字幕在视频播放时在输出窗口和 Flash 文件文本区域中都可以显示出来。

5.3 在 ActionScript 音频声道应用

USM 可以为多音频声道编码，使视频包含独立的声道应用于不同的语言。在 SWF 中访问这些音频声道过程非常简单，只需要很少的代码。

访问视频文件的音频声道，一个 Flash SWF 必须创建一个拥有/播放一个至少包含单通道的 USM 视频文件。该过程的简要介绍，请参考 Flash 中视频添加和视频音频通道编码相关章节。

在 *mvplayer* 动画剪辑中放置以下代码到 *actions* 图层的帧 1 中。输入或者拷贝代码到 *Actions* 图层面板的末尾。

1. 如果尚为准备好，双击 *mvplayer* 动画剪辑输入时间轴。

2. 在 *mvplayer* 内部，选择时间轴 *actions* 图层第一个关键帧，输入代码段到 *Actions* 面板，位于所有已有代码之后。

3. 声明两个变量。第一个 `AudioTracks` 为一个包含每个音频通道的数组。第二个， `CurAudioTrack`，将保存当前选择通道，初始化这个变量为 `0`。

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

4. 下一步，设置回收函数来监听视频元数据。分配视频 `audioTracks` 元数据给 `audioTracks` 数组。

   注释：如果‘`ns.onMetaData`’在增加字幕控制小节已经存在，在代码块内部增加代码行—‘`AudioTracks = infoObject.audioTracks;`’—代码添加在代码行‘`trace("Subtitle Channels: " + subtitleChannelNumber);`’后面，在结束大括号‘}’前面。

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

5. 为测试用，在 *mvplayer* 动画剪辑中的场景里创建一个简单的正方形图动画剪辑，可以当作为一个按钮。设置动画剪辑实例名称为 ‘prev_audiotrack’ 。

6. 在‘mvplayer’中‘action’图层中的第一个关键帧中输入以下代码到 *Actions* 面板。这些代码在 next_audiotrack 按钮有效时能够循环播放音频通道。

```
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* 的场景中从步骤 5 复制动画剪辑，取其实例名称为 ‘next_audiotrack’ 。

8. 在‘mvplayer’中‘action’图层中的第一个关键帧中输入以下代码到 *Actions* 面板。这些代码在 next_audiotrack 按钮有效时能够循环播放音频通道。

```
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 中发布并测试 Flash 文件。如果所有步骤执行正确，播放视频的各种类型的音频声道可以预览，只需点击 *Next* 和 *Previous* 音频按钮（如果编码的 USM 文件包含多声道；只能播放单个声道）。

6 ActionScript 视频扩展

在支持 Scaleform 的 Flash 中有许多 ActionScript 方法用来播放视频文件。另外，Scaleform 还有一些新的扩展专用功能。搜索关于‘NetStream’的 Adobe 帮助文件可以查看更多关于内嵌 NetStream 属性、方法和事件的信息。

6.1 支持内嵌 NetStream 属性

`currentFps:Number`

每秒显示的帧数。

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

`time:Number`

.播放进度，以秒为单位。

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

6.2 支持内嵌 NetStream 事件

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

播放 USM 文件时当到达内嵌播放暗点位置时调用。

注释： 在 Adobe Flash Help 中搜索‘onCuePoint’获得更多信息。

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

USM 文件播放时当播放器接收到内嵌描述信息时调用。

注释： 在 Adobe Flash Help 中搜索‘onMetaData’获得更多信息。

Scaleform Extensions:

`audioTracks`

USM 文件中音频声道数组。

例子：

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

subtitleTracksNumber

USM 文件中字幕通道总数。

例子：

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

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

每次 NetStream 对象的状态改变或者错误消息提示时调用。

注释： 在 Adobe Flash Help 中搜索‘onStatus’获得更多信息。

6.3 支持内嵌 NetStream 方法

```
close()
```

停止播放所有流数据，设置 ns.time 属性为 0，使流数据可被另外部分使用。

例子： `ns.close();`

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

暂停或者继续流媒体播放。

例子： `ns.pause(true);`

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

开始播放外部视频（USM）文件。

例子： `ns.play("myVido.usm");`

`seek(offset:Number)`

搜索离流媒体开始处几秒内最近的的关键。

例子： `ns.seek(50);`

`setBufferTime(bufferTime:Number)`

以秒为单位设置将要缓冲的电影数据的数量。GFx::Video::Video 从磁盘缓冲足够的数据，以实现流畅播放并减少磁盘读取次数。缓冲区大小以视频位率 (Bitrate) 以及其他视频参数为依据。默认情况下，此缓冲区大小将足以容纳 1 秒播放。

示例： `ns.setBufferTime(2.5);`

6.4 新 Scaleform 属性

`subtitleTrack = Number`

NetStream 中当前使用的字幕通道数。

例子： `ns.subtitleTrack = 1;`

`audioTrack = Number`

NetStream 使用的主音频轨道。当视频文件有多个音频轨道（例如，一个英语版本和一个西班牙语版本）时，主音频轨道可用来针对特定语言设置音频轨道。

示例： `ns.audioTrack = 2;`

`subAudioTrack = Number`

一个次要或 SubAudio 轨道，一般由 NetStream 用来随电影一起播放对话轨道或音效。

示例： `ns.subAudioTrack = 3;`

`voiceTrack = Number`

取代 5.1ch 音频的中心轨道。仅使用此来更改 5.1ch 音乐的语音轨道。

示例： `ns.voiceTrack = 1;`

`setReloadThresholdTime(reloadTime:Number)`

设置重新加载时间。确定从磁盘重新填充视频数据缓冲区的频率。例如，如果一个应用程序将缓冲区大小设置为 4 秒（通过 setBufferTime），而把重新加载阈值 (Reload Threshold) 设置为 1 秒，则 GFx::Video::Video 将会先用 4 秒长的数据填充缓冲区。在 3 秒长数据解码并使用后，就剩下不到*重新加载阈值*秒长的数据，而 GFx::Video::Video 就会重新填充缓冲区。

示例： `ns.setReloadThresholdTime(1);`

`setNumberOfFramePools(numPools:Number)`

设置内部视频缓冲区的数目。GFx::Video::Video 使用内部内存在显示前缓冲已解码的帧。较多帧池有助于在高 CPU 负载下理顺播放活动。默认值为 2。

示例： `ns.setNumberOfFramePools(3);`

`setOpenTimeout(timeout:Number)`

以秒为单位设置打开文件超时（USM 头解码所必需的时间）。默认值为 5 秒。要禁用此功能（设置无限超时），请传递 0。

示例： `ns.setOpenTimeout(5);`

`currentFrame:Number`

返回当前播放位置（以帧计）。

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

`Loop = Boolean`

设置循环为 true 告知 NetStream 循环播放视频，直到循环属性设置为 false。

例子： `ns.loop = true;`

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

用来显示视频的一个给定区域。

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

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

当播放器接受到一个来自 USM 文件的字幕字符串时调用。

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

7 视频创建理解

视频编辑或者视频创建可以为一个非常有技巧的过程，特别是对于那些对于视频编辑和转换没有太多经验的人来说尤其如此。视频编辑包含了很多选项和设置，都将对视频播放质量和性能产生影响。对于视频，典型的需要考虑分辨率、纵横比、比特率、帧速率和其他相关因素，使其能够以最佳质量画面质量流畅播放。每个硬件平台，使用案例和游戏引擎在视频编码和保证高性能方面使用的方法略有不同。这需要在编码过程中做一些实验和测试。

例如：播放单个全屏视频动画需求较为简单，因为作为唯一运行程序可以利用系统大多数资源。但是，当需要用到多媒体流动画、导入动画背景、使用 alpha 效果、在纹理上播放动画或者执行其他高级功能，则需要做一些不同方法的实验来确定正确的设置。

当创建一个完整的 1080p 视频可能是在 Xbox 360 和 PS3™上创建高质量动画最好的方法，创建视频之前对一些因素必须进行评估。首先，编辑 HD 视频，因为文件尺寸非常大，就会极其复杂且速度也会很慢。HD 视频，特别是高比率率 1080p 视频看起来效果非常好，但是文件很大，播放时占据很多系统资源，所以必须将每个视频中需要使用的分辨率确定下来。

是否需要创建一个 1080p 的视频动画，其他需要考虑的因素为：视频源文件为什么（实时动画、CG、游戏内部画面等）、视频创作人、预算和开发周期。

非常重要的是检查在 720p、1080p 和其他 HD 分辨率中实时画面质量的区别，通常情况下低分辨率画面看起来已经能满足需求，且创建容易、文件较小（不会占据很多硬盘空间）、容易编辑、动画流更加高效且播放占据更少内存空间。在创作过程中评估和测试非常重要，根据实际要求确定工程目标，不单是考虑视频的播放情况，还要考虑视频流和创建过程是否高效。

最好是设定视频为最终理想的分辨率或者缩小规模。Scaleform 视频编码器能够调整视频大小，但是可能会损失质量、画面拉伸或者其他因素。这在对低分辨率视频进行扩展时特别明显；但在对高分辨率视频进行缩小播放时影响很小。

几乎任何的视频编辑软件都可以与 Scaleform 视频配合使用。为达到更好效果，视频必须为未经压缩的 AVI 文件。使用 Scaleform 视频文件压缩时需要使用未经压缩的文件。如果 AVI 文件由另外的编码器创建（如 DIVX）并且进行了视频压缩（进行艺术加工），这样，将视频文件转换成 Scaleform 视频格式时相当于进行了两次压缩，这样不能减少视频文件大小，反而会影响视频质量。

创作视频时，音频也是一个重要因素。需要知道是否需要支持单声道、立体声或者 5.1 声道；但是，音频定位可能更为重要。需要考虑配音文件如何创建，特别是当需要为每种不同语言翻译和录音，准备一个配音文件（不单是创建字幕）。Scaleform 视频能够采用多声道音频轨道，可以用来为不同语言播放不

同配音，这样可以在单个视频文件中支持多种语言，而无需创建多个视频文件包含单独的语言。但是，使用这种方法，配音内容需要从视频/音频编辑程序中导出为单独的文件，这样才可以在运行中进行交叉调用。

再次强调，在初始化视频作品前，最好在整个开发过程中考虑到所有的游戏视频需求和性能的测试指标，以避免时间浪费或者长远的性能问题。

7.1 Alpha 通道

对于视频编辑和电脑图像不熟悉的用户可能对 alpha 通道或者蒙版概念不是很熟悉。Scaleform 视频支持像素点 alpha 通道，意思为视频中的每个像素点可以为实体颜色或者是一定程度的透明化显示。在一个典型的视频编辑程序中，视频可以为完全或者部分透明，这根据分配在通道上的控制透明程度的颜色而定。

这在通常的 TV 和电影里使用的“绿光屏”中经常用到，其中演员以绿色实体（或者蓝色）为背景，这样可以易于删除背景，将演员合成到其他场景中。亮绿色或者蓝色用的最多，因为这两种颜色最显眼，且在电影的其他画面中很少用到，所以不容易与银幕其他画面混淆。视频画面能够以全透明或者部分透明编码。

在大多数程序中增加一个 alpha 通道相对容易，增加 alpha 效果到视频编码犹如点击一个检验按钮那么容易。但是，需要注意的是具有 alpha 效果的视频在渲染上比没有 alpha 效果要复杂两倍。在游戏中需谨慎使用 alpha 透明效果，特别是在实时游戏引擎中。

8 技术集成指南

本章提供了集成 Scaleform 视频到你所用的引擎需要了解的技术要点。

8.1 Scaleform 声音系统初始化

初始化 Scaleform 声音系统，需要将一个 GFx::Audio 类实例设置到 GFx::Loader 对象上。该状态的目的为提供声音渲染对象，SoundRenderer 和播放 SWF 声音流同步参数，SoundRenderer 为一个抽象的 C++ 接口，应该在游戏中用来产生声音效果。Scaleform 默认为需要执行，该组件基于 FMOD 跨平台声音库并可以在所有支持的平台上使用，要创建一个该实例，需要调用 Sound::SoundRendererFMOD::CreateSoundRenderer 方法，然后创建的对象应该用 FMOD::System 对象初始化。

例子：

```
// 初始化声音渲染对象
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);
Ptr<Sound::SoundRenderer> psoundRenderer =
                          *SoundRendererFMOD::CreateSoundRenderer();
if (!psoundRenderer->Initialize(pFMOD))
      exit(1);

// 在导入 GFx::Audio 上设置一个实例
Ptr<GFx::Audio> paudioState = *new GFx::Audio(psoundRenderer);
loader.SetAudio(paudioState);
```

8.2 视频播放系统初始化

需要初始化 Scaleform 视频播放系统，Video 类的一个实例应该设置到 GFx::Loader 对象。本状态的目的为当应用程序需要播放视频文件时创建一个视频播放器的实例。如果视频文件包含音频文件并需要播放，则 VideoSoundSystem 接口的一个实例需要被设置到 GFx::Video::Video 对象。Scaleform 为每个支持的平台提供了可执行的接口，就是以执行为基础的 SoundRenderer 接口。

例子：

```
// 导入中设置一个 GFxVideo 实例
Ptr<GFx::Video::Video> pvc = *new Video(Thread::NormalPriority);
// 设置基于 SoundRenderer 接口的视频音频系统实例 instance which is based on
SoundRenderer

if (psoundRenderer)
{
      pvc->SetSoundSystem(psoundRenderer);
}
// 设置一个应用于特殊平台的视频音频系统
// 注意：SetSoundSystem 方法应该在每个 Video 中调用一次。
pvc->SetSoundSystem(Ptr<Video::VideoSoundSystem>(*new
                                    VideoSoundSystemDX8(0)));

loader.SetVideo(pvc);
```

在 Windows 平台，Video::Video 类叫 VideoPC 然后需要三个参数。第一个参数是视频译码线的优先。第二个参数控制用多少个线来做视频译码，然后第三个参数是每一个译码线的 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);
```

对于 PS3 和 Xbox360 平台，Scaleform 提供了特殊的 Video 类，包含一些额外的初始化参数。

对于 PS3，本状态命名为 GFx::Video::VideoPS3 并包含 5 个参数：

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

spurs -	CellSpurs 结构指针
spu_num -	用于视频编码的 SPU 数量
ppu_num -	用于视频编码的 PPU 线程数量
ppu_priority -	PPU 线程优先级
spurs_priorities -	SPU 核的优先

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 占用率示例：

可以初始化 Scaleform Video 来实现其 PPU 占用率最小化。例如，可以用 50-60 FPS 播放 720p 视频，PPU 占用率不到 5%。当主线程忙到足以达到抢占视频解码线程的程度时就会发生这种情况。例如，假如某个游戏正在运行,主线程就会忙到足以造成这种情况的程度。

请牢记，PPU 对解码线程的占用情况取决于是否加载其他线程。如果主线程不忙，解码线程就只占用大约 40% 的 PPU。但在以下设置的情况下，假如主线程忙，它就会优先于解码线程，而解码线程就只使用 4-5% 的资源。事实上，假如优先级设置正确，解码线程的占用率可低至 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);
```

对于 Xbox360 而言，该状态被命名为 VideoXbox360，并使用三种参数来规定使用哪种硬件线程进行视频解码。第一个参数规定了两个线程，它们主要完成大部分内部工作，因此这些线程不应重叠。第二个参数控制译码视频线的优先。第三个参数控制视频渲染器的硬件线。

```
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 背景游戏数据加载API

在视频回放时，游戏数据的背景加载将通过回调界面 Video::VideoBase::ReadCallback 和 – VideoBase::IsIORequired、 VideoBase::EnableIO 等方法来执行。

ReadCallback 是一个回调界面，它的作用是在要求视频进行读取操作和完成时对该程序进行通知。

例子：

```
// 视频系统的简单读取回调运行
class VideoReadCallback : public VideoBase::ReadCallback
{
      public:
      VideoReadCallback() {}
      virtual ~VideoReadCallback() {}

      virtual void OnReadRequested()
      {
          // 视频读取开始
          VideoReadStart = Timer::GetTicks();
          // ...
      }
      virtual void OnReadCompleted()
      {
          // 视频读取完成
          GameReadStart = Timer::GetTicks();
          // ...
      }

      UInt64  VideoReadStart;
      UInt64  GameReadStart;
};

// 初始化视频系统
Ptr<Video::Video> pvc = *new Video::VideoPC(
```

```
                    Thread::NormalPriority, MAX_VIDEO_DECODING_THREADS, affnts);

    // 创建并设置视频读取回调
    Ptr<VideoReadCallback> pvideoReadCallback = *new VideoReadCallback;
    pvc->SetReadCallback(pvideoReadCallback);
```

IsIORequired 确定了视频系统是否需要运行任何视频数据读取操作。只要这一方法的返回值为真，该程序就应当立即停止所有磁盘输入输出操作，并通过调用 *EnableIO*.来启用视频读取操作。

EnableIO 为有效数据的背景加载启用/禁用视频读取操作。

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");
    }
```

视频背景加载功能将通过两个演示（VideoBGLoadFlash 和 VideoBGLoadData）来说明，它们的完整源代码可在 *Apps\Samples\Video* 目录中获取。

- VideoBGLoadFlash – 这一演示说明了当某一视频播放时，在您的程序中加载 Flash 内容的简单运行过程。
- VideoBGLoadData – 这一演示为播放视频时如何向程序中加载任意数据提供了范例。

8.4 Scaleform VideoSoundSystem 界面

GFx::Video::VideoSoundSystem 是一个抽象接口，为 Video::Video playback 回放提供音频支持；开发者通过对这一类作出自己的版本来替代音频运行。在播放视频之前，需要为这一类创建一个实例，并安装 Video::SetSoundSystem() 。通常情况下，特定平台的运行通常能够避免运行这一界面。

音频系统界面包含在 Video 分布中：

- Video::VideoSoundSystemDX8 – Windows 中的 DirectSound
- Video::VideoSoundSystemXA2 – Windows 与 Xbox360 中的 XAudio2
- Video::VideoSoundSystemPS3 – MultiStream for PS3
- Video::VideoSoundSystemWii – Wii 系统音频
- Video::VideoSoundSystemFMOD - 基于 FMOD 的音频界面
- Video::VideoSoundSystemWwise – 基于 Wwise 的音频界面

在当前版本的 Scaleform 中，由于嵌入了 Flash 音频回放，视频与音频支持是解耦的，允许在使用视频时，不必要求一般的音频引擎。为了完成这一工作，视频将通过一个独立的 Video::VideoSoundSystem 类获得支持，它与 Scaleform 中的其余 Sound::SoundRenderer 是分开的（详情请参阅第 8.1 节内容）。这意味着未来获得视频支持，你只需运行 VideoSoundSystem 和 VideoSound 类，这比 SoundRenderer 要简单很多。请注意，如果你已经运行了 SoundRenderer，你可用它对 Video 进行直接初始化，因为它提供了一个功能集。在某些情况下，你还可以对二者进行混合运行（如果自定义视频音频类比一般音频引擎提供了更好的流媒体支持，那么这一做法是很有帮助的）。

例子：

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

对于视频音频支持而言，需要运行 GFx::Video::VideoSoundSystem 和 GFx::Video::VideoSound 。通常而言，在视频初始化过程中，仅安装 GFx::VideoSoundSystem 的一个实例。GFx::VideoSoundSystem 提出了一个单一方法， Create，用于创建 VideoSound 对象，代表独立的视频音频流。当有新的视频被打开后，Scaleform 就会调用这一函数（可能会同时播放多个视频）。当每一个 VideoSound 对象创建完成后， Scaleform 将会调用各种函数来说明开始和停止音频输出。实际的音频数据将通过 VideoSound::PCMStream 轮询后传递给指定音频后获得。轮询功能通常由一个单独的线程来完成，该线程由 VideoSoundSystem 维护，从而为激活的音频服务。

请注意，基于音频输入插件的 VideoSoundSystemWwise 可作为 Wwise SDK 的一部分来运行，因为 v2009.2.1 build 3271. Audiokinetic 提供了完整的源代码和 Visual Studio 方案/项目。这一插件放置的位置

为： *SDK\samples\Plugins\AkAudioInput.*。详情请参考 Wwise 文档。 GFx::Video::Video 分布不包括 Wwise SDK 的任何部分，并且应当单独安装。

例子：

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

更多详情请参考 Scaleform Player - FxPlayerAppBase.cpp、 FxSoundFMOD.cpp、 FxSoundWwise.cpp 等源代码，这些源代码的位置是 *Apps\Samples\FxPlayer* 和 *Apps\Samples\Common* 目录，并运行音频系统界面 Video_VideoSoundSystem*.cpp ，其位置是 *Src\Video* 目录。

8.5 多语言视频

标准的多语言视频每种语言有多个音频轨道。GFx::Video::Video 采用 ActionScript 视频扩展，使用户可以有效地处理多语言电影数据。例如，我们来看一个同时包含英语和西班牙语立体声音频的视频。播放此视频时，可以为每种语言选择一个音频轨道。要选择音频轨道，请使用 NetStream 提供的 *audioTrack* 扩展。

示例： `ns.audioTrack = 2;`

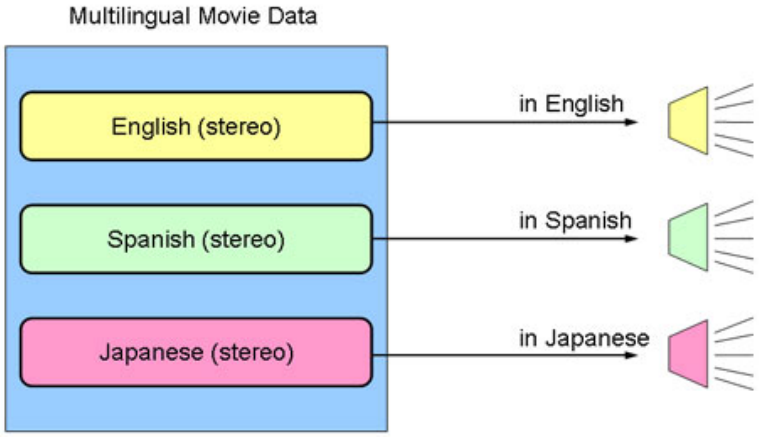

图 12：多语言视频

多语言环绕声视频是拥有一个通用 5.1 音频轨道和多个语言轨道的视频。尽管语言轨道可随环绕声一起切换，但只能将一个环绕声轨道与多个语言轨道一起使用。这种在多个环绕视频中共享单个声音轨道的做法减小了电影大小。

GFx::Video::Video 支持以下两种不同的播放功能：

1. 中心通道替换
2. SubAudio 播放

8.5.1 中心通道替换

如上所述，多语言环绕声视频减小了电影文件大小以支持多种语言。多语言环绕视频针对每种语言有一个通用的 5.1 音频轨道和若干单声道语音轨道。通常情况下，播放单声道语音轨道，而不是 5.1 通道音频的原始中心轨道。当播放多语言环绕声视频时，需要单独选择 5.1 音频轨道和单声道语音轨道。
如果要从中心通道播放音乐或声音效果，必须提前将这些与语音轨道混合在一起。
示例：

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

如果主轨道不是 5.1 音频，或者语音轨道不是单声道，就会忽略语音轨道的设置。如果将同一主音频轨道设置为语音轨道，正常情况下会播放中心通道。

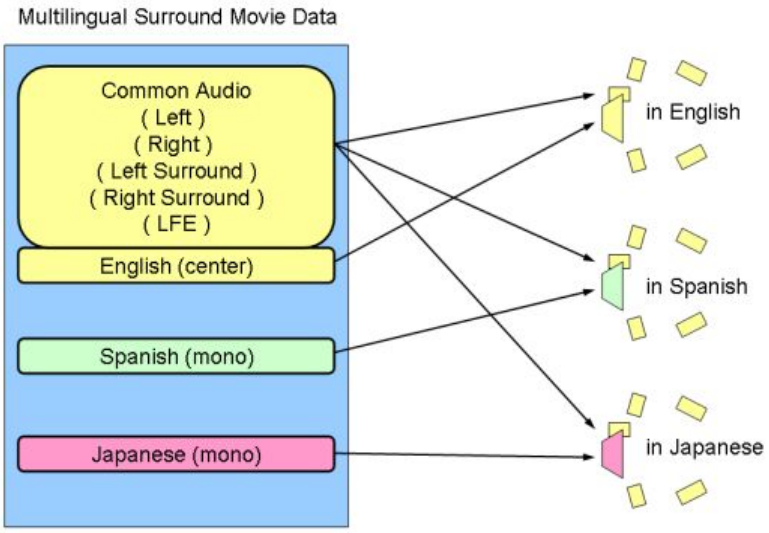

图 13：中心通道替换

8.5.2 SubAudio 播放

使用 SubAudio 播放功能时，应用程序可以通过 GFx::Video::Video 同时与主轨道一起播放另一个轨道。确切地说，GFx::Video::Video 通过一个附加声音接口以及主轨道来输出声音数据。因此，应用程序可以通过作为语音轨道输出 SubAudio 轨道来执行多语言环绕声播放。

示例：

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

请关注： SubAudio Playback 能让您在主音轨和副音轨里独立控制音效声量。 只需要在里用 Sound.setVolume() 扩展语法就能控制音效.

示例：

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

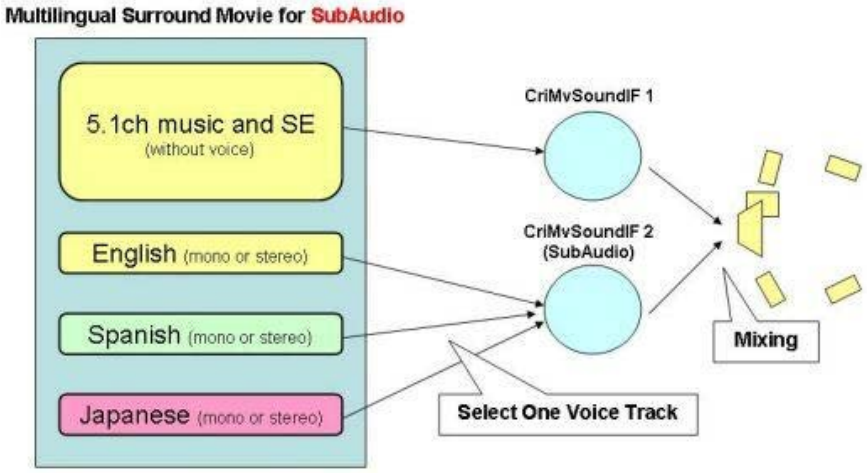

图 14：SubAudio 播放

9 故障排除

如果您的视频内容播放不流畅，而且您正在遭遇时断时续 (Stuttering)，这通常是因为硬件没有足够可用资源来充分处理视频数据。一般情况下，这可以通过执行以下操作进行修补：

1. 释放您的代码中的资源（即确保播放视频时处理或 IO 负载不重），或者

2. 减小视频数据的大小和比特率，或者

3. 向视频系统提供更多资源（线程、帧池）。

具体说来，下面介绍三件事情（优先级依次降低），通过调整这三件事情来帮助确保视频播放流畅：

1. *线程配置* - 更改数量和优先级。看看您的游戏引擎已经在用哪些线程，或者哪些尚未使用。较高分辨率视频（720P 以上）一般需要 3 到 4 个线程。有关配置线程的更多信息，请参阅上面第 8 节。

2. *视频分辨率和比特率* - 您可能需要减少您的视频的大小和/或比特率以减轻工作负载。Scaleform 将会根据需要自动按比例增加低分辨率视频以适应视窗。

3. *缓冲器和帧池* - 许多 ActionScript 方法和扩展可用于配置视频播放。下面的两个调用可用来调节视频系统使用的内存缓冲时间。有关更多详细信息，请参阅上面第 6 节。
   a. `NetStream.setNumberOfFramePools(numPools:Number)`
   b. `NetStream.setBufferTime(bufferTime:Number)`