Autodesk®
Scaleform®

스케일폼 비디오 시작하기

이 문서는 Scaleform 4.3 스케일폼 비디오를 사용하는 방법에 대하여 설명한다.

집필: Matthew Doyle, Vladislav Merker

버전: 3.01

최종 편집: 2011 년 9 월 15 일

Autodesk®
GAMEWARE

Copyright Notice

Autodesk® Scaleform® 4.3

Movimento, MPA, MPA (design/logo), MPI (design/logo), MPX, MPX (design/logo), Mudbox, Multi-Master Editing, Navisworks, ObjectARX, ObjectDBX, Opticore, Pipeplus, Pixlr, Pixlr-o-matic, PolarSnap, Powered with Autodesk Technology, Productstream, ProMaterials, RasterDWG, RealDWG, Real-time Roto, Recognize, Render Queue, Retimer, Reveal, Revit, Revit LT, RiverCAD, Robot, Scaleform, Scaleform GFx, Showcase, Show Me, ShowMotion, SketchBook, Smoke, Softimage, Socialcam, Sparks, SteeringWheels, Stitcher, Stone, StormNET, TinkerBox, ToolClip, Topobase, Toxik, TrustedDWG, T-Splines, U-Vis, ViewCube, Visual, Visual LISP, Vtour, WaterNetworks, Wire, Wiretap, WiretapCentral, XSI.

Disclaimer

연락처:

문서	Getting Started with Scaleform Video
주소	Scaleform Corporation
	6305 Ivy Lane, Suite 310
	Greenbelt, MD 20770, USA
웹사이트	www.scaleform.com
이메일	info@scaleform.com
직통전화	(301) 446-3200
팩스	(301) 446-3199

목차

1 환영인사

CRI™에 의해 강화된 Scaleform Video 는 Scaleform™와 완벽하게 통합된 비디오 솔루션이다. Scaleform Video 는 사용자가 고성능 비디오 재생능력을 플래시의 애셋들을 사용해서 플레이 할 수 있게 해주며, 플래시 비디오 파이프라인에 통합해 준다. 새로운 비디오 모듈을 사용하면 고 정밀도의 잡음 없는 비디오를 로고, 메인 메뉴, HUD, 게임텍스처, 전체화면 컷씬(cut scene) 등의 용도로 사용할 수 있다.

CRI 무비 코덱은 기존 코덱들보다 재상과 인코딩에서 장점이 있어서 선택되었다. CRI 무비의 차세대 재생 엔진은 실시간 게임 시스템을 위해서 특별히 만들어졌으며, 최신 멀티 코어 하드웨어를 활용한다. CRI 미들웨어는 1700 개 이상의 게임에서 사용 중이다.

Scaleform Video 를 사용하면 PC 와 게임콘솔(X360, PS3, Wii)에서 전체화면, 창모드, 알파투명도 등을 지원하며, 어떠한 해상도의 동영상도 재생 가능하다. 또한 스트리밍과 멀티 쓰레드 지원용으로 밑바닥부터 설계되어있다. Scaleform Video 를 사용해서 서로 다른 해상도의 비디오를 각 플랫폼에 맞게 최적화, 프레임 레이트, 해상도, 비트레이트, 크기, 화면비율 등을 조절 함으로써 최적의 영상과 성능을 보장받을 수 있다.

Scaleform Video 의 워크플로우는 사용자로 하여금 어도브 프리미어, 애프터 이펙트 등의 어떠한 비디오 편집 프로그램에서도 손쉽게 만들수 있게 해주는데, 동영상을 플래시로 임포트하고, 게임으로 쉽게 빠르게 내보낼(export) 수 있게 해준다. 이러한 작업 중에도 자막을 붙이거나, 5.1 오디오를 추가하거나, 다른 언어로 목소리를 더빙하거나 하는 등의 게임 내의 무비들에 다양한 상호작용과 관련된 내용을 넣을 수 있다. Scaleform Video 는 다양한 진보적 기능을 지원하는데, 알파채널, 인터페이스나 컷씬(cut scene) 이상의 용도로 사용할 수 있게 해주는 큐포인트(cue point) 등이 그것이다.

Scaleform Video SDK 는 다음을 포함한다:

- Scaleform Video 런타임 라이브러리
- Scaleform Video 툴
- 학습자료/샘플
- Scaleform Video 시작하기(이 문서)

2 스케일폼 비디오에 관하여

스케일폼 비디오는 고화질, 고음질 비디오 재생 시스템이며 다양한 플랫폼을 지원한다. 또한 플랫폼의 특징을 최대한 활용하여 글로벌 표준의 고급 무비를 만드는 것을 돕느다.

2.1 설치위치

스케일폼 비디오 인코더는 다음 위치에 설치된다:

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

스케일폼 비디오 데모는 다음 위치에 설치된다:

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

이 데모에는 3 개의 다른 비디오 플레이어가 서로 다른 파일을 하나의 윈도우에 출력하는 것이 포함되어 있다. 이 디렉토리는 또한 데모와 관련된 몇 개의 플래시 파일과 비디오 파일도 포함되어 있다. 이 파일들은 테스트를 위하여 미리 인코딩 된 USM 파일이 포함되어 있다. 데모를 돌려보려면 Scaleform 플레이어에서 videodemo.swf 를 실행해보기 바란다.

2.2 학습자료 파일

인코더가 있는 동일한 디렉토리에는 학습용 샘플 파일이 있는데, 이들을 다음과 같은 용도로 사용하라.

- *getting_started_with_video_tutorial.fla* – 이 학습예제에서 사용되는 모든 것이 포함된 원본 파일
- *getting_started_with_video_tutorial.swf* – 플래시 배포파일
- *sample.avi* – 인코딩용 샘플 비디오 파일
- *sample_audio.wav* – 인코딩용 샘플 오디오 파일
- *sample_cue_points.txt* – 인코딩용 샘플 큐포인트 파일
- *sample_subtitles.txt* – 인코딩용 샘플 자막

2.3 인코더 디렉토리

스케일폼 인코더 디렉토리에는 작업에 중요한 몇 개의 파일들이 존재한다.

- Medianoche.exe — 명령행 인코더
- ScaleformVideoEncoder.exe — 인코더의 그래픽 외피(껍데기)
- TMPGLib.dll — 인코더의 심장부
- VideoEncoderUtil.dll —ScaleformVideoEncoder.exe 에서 필요한 라이브러리
- VideoPlayer.swf —스케일폼 인코더 윈도우의 미리보기(Preview)가 눌렸을 때 출력물을 제어하기 위한 비디오 플레이어

2.4 장점

- **고급 무비 재생**
  간단한 API 를 사용해서 고급 무비를 재생하는 것이 가능함. 최고사양의 HD(1080p) 영상 재생도 가능.

- **다중언어 및 자막 지원**
  다중언어와 자막이 하나의 무비에 들어감. 다양한 지역(혹은 국가)에서 판매될 게임을 개발할 때 개발비용 감소.

- **쉬운 조작과 전문가 수준 인코딩**
  사용하기 간편한 인코딩 툴이 SDK 패키지에 포함되어 있음.

2.5 기능

CRI Movie 로 강화된 스케일폼 비디오는 개발자로 하여금 플래시를 사용한 쌍방향 컨텐츠를 높은 성능으로 재생할 수 있도록 해주는 프리미엄 모듈이다. 스케일폼 비디오를 사용하면, 고급, 고화질, 잡음 없는 영상을 다양한 차세대 플랫폼에서 재생할 수 있다. 동영상 재생이란 전체화면, 창모드, 3D 게임엔진의 텍스처에, 반투명을 지원하면서 재생하는 것을 지원한다는 말이다. 다양한

애플리케이션을 위한 쌍방향 컨텐츠에 동영상을 사용할 수 있는데, 인트로 로고, 메인 메뉴, 게임 HUD, 게임 텍스처, 게임 비디오 스크린, 전체화면 컷씬, 로딩 화면 등등이 그것이다. 스케일폼 비디오는 이미 Scaleform 4.1 과 완벽하게 통합되어 있기 때문에 게임내의 비디오 재생에는 거의 추가적인 작업이 필요 없다. 스케일폼 비디오는 유명한 CRI Movie 코덱에 기반하고 있기 때문에 엄청난 성능 향상을 얻을 수 있을 것이다.

2.6 기술적 하이라이트

- **비디오와 오디오 포맷:** 가장 일반적으로 사용되는 비디오, 오디오 포맷인 AVI 와 WAV 가 인코더에서 지원됨
- **멀티 플랫폼 지원:** 스케일폼 비디오는 모든 메이저 플랫폼인 XB360, PS3, PC, Wii 를 지원하므로 프로그래머를 하드웨어 사양에 의존적인 개발에서 해방시켜줌
- **다른 플랫폼용 최적화 인코딩:** 인코딩 중에 최적화 인자값을 조절해서 플랫폼의 특징을 반영한 비디오 퀄리티/압축률 등을 조절
- **자막:** 동일한 동영상용 다중 자막 트랙이 지원된다. 이렇게 되면 사용자가 분리된 파일에 자막 트랙을 만들 수 있으며 또한, 각 자막은 각기 다른 시작시간과 엔딩타임을 가질 수 있다. 다중 언어 문자는 자동적으로 Scaleform 폰트설정 시스템에 의해서 적절하게 지역화 된다.
- **알파채널 지원:** 사용자는 "가상 투명도"를 사용해서 동영상에 픽셀별 투명도로 내장 알파를 지정하거나, 동영상 전체 투명도를 지정할 수 있다. 이를 이용해서 다양한 효과를 만들어 낼 수 있다.
- **텍스처에 렌더:** 동영상이 게임 텍스처에 렌더링 될 수 있기 때문에 사용자가 동양상을 게임에서 직접 활용하는 것이 가능하다.
- **오디오 트랙:** 동영상 하나 당 다중 오디오 트랙(최대 32 개)을 지원한다. 이를 이용해서 다중 언어 음성을 지원할 수 있다.
- **오디오 트랙 전환:** 액션스크립트나 Scaleform 4.3 플래시 확장을 사용해서 오디오 트랙간에 전환이 가능하다.

- **지역화 지원:** 오디오 트랙과 자막 트랙을 실시간에 액션스크립트에서 전환할 수 있기 때문에 결과적으로 스케일폼 비디오에서 지역화를 지원하는 셈이다. 이를 사용하면 UI 동영상을 다른 언어로 음성 지원하도록 구성하는 것이 매우 쉬워진다.
- **서라운드 사운드 지원:** 각 오디오 트랙은 모노/스테레오/5.1 채널 서라운드를 지원할 수 있다.
- **다중언어 서라운드 사운드:** 서라운드 사운드에서 센터채널을 보이스(목소리)로 교체 할 수 있다. 이렇게 하면 서라운드 오디오 데이터에서 음성을 손쉽게 출력할 수 있다.
- **탐색 능력:** Scaleform Video 를 사용하면 지정된 시간 값을 비디오 스트림에서 찾을 수 있다. 이러한 값은 스트림의 시작점과 현재 값이다. 사용자는 양수나 음수 값을 사용해서 되감거나(rewind), 앞으로 진행(forward)할 수 있다.
- **큐포인트:** 큐포인트란 비디오 클립에서 매우 중요한 시간 값을 말한다. 큐포인트를 사용하면 비디오 클립내의 다른 세그먼트에 접근 할 수 있다. 스케일폼 비디오 인코더를 사용해서 USM 파일에 직접적으로 큐포인트를 내장시킬 수 있다. 이렇게 내장된 큐포인트는 플래시에서 사용된다. 비디오 내의 내장 큐포인트에 도착하게 되면 액션스크립트의 방아쇠(트리거)가 당겨져서 그 큐포인트와 연관된 모든 정보들이 사용자에게 전달된다. 이를 사용해서 네비게이션 메뉴를 설정하거나, 비디오의 특정 이벤트를 탐색해서 쌍방향 비디오를 만들 수도 있다.

2.7 파일 재생

스케일폼 비디오는 비디오 무비를 스케일폼 CRI 비디오 포맷인 USM 파일로 변경한다. USM 파일을 재생하려면 간단하게 데스크탑에 있는 Scaleform Media Player 에 파일을 끌어놓기 하며 된다. 혹은 스케일폼 플레이어를 구동해서 열어도 상관은 없다.

실제로 C:\Program Files\Scaleform\GFx SDK 4.3\Bin\Data\AS2\Video\scaleform_logo.usm 파일을 플레이어에서 재생해 보도록 하자.

2.8 워크플로우

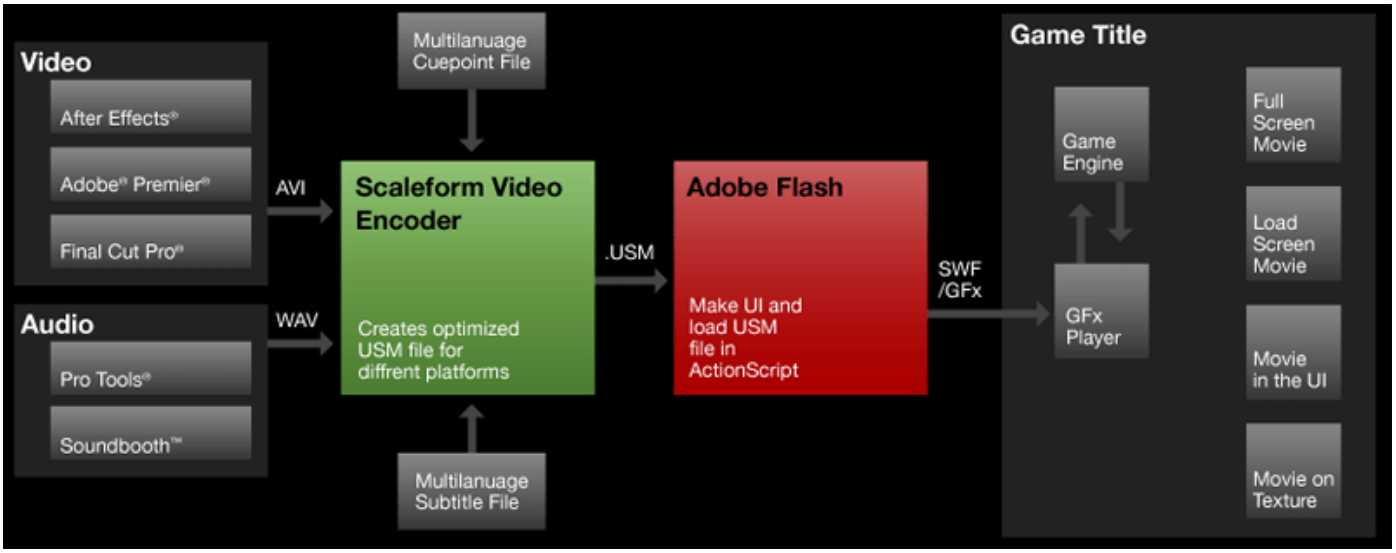

그림 1: 스케일폼 비디오 워크플로우

워크플로우는 게임 내 동영상을 다음과 같이 생성할 필요가 있다.

1. 스케일폼 비디오 API 를 게임엔진과 통합
2. 어도브 프리미어나 기타 동영상 인코더를 사용해서 AVI 를 제작
3. 스케일폼 인코더를 사용해서 AVI 를 USM 으로 변환
4. USM 인코딩된 비디오를 사용할 플래시 SWF 파일 생성
5. 비디오, 자막, 큐포인트 등을 제어하는데 사용할 액션스크립트 작업
6. 만들어진 SWF 파일을 게임에서 읽어오기
7. 게임엔진에서 사용 가능한 방법으로 SWF 파일 재생

주의: USM 비디오 파일을 플래시 파일에 내장하려 하지 말고, 액션스크립트를 사용해서 참조하라. 일단 비디오가 설정되고 게임에서 한번이라도 재생이 되면 반복 재생을 할 때는 파일에 대한 아무런 업데이트가 필요 없다. 하드디스크의 동일한 위치에 동일한 파일명으로 존재한다면 말이다.

3 시작하기: 스케일폼 비디오 인코딩

Scaleform 4.3 에서 사용되는 비디오는 CRI 의 USM 포맷으로 인코딩 되어야 한다. CRI 포맷은 고퀄리티의 HD 디스플레이 출력을 위한 게임에 최적화 되어 있다. 이를 위해서 스케일폼 인코더를 제공한다. 이번 장에서는 USM 인코딩과 큐포인트 포함, 자막, 오디오 통합 등에 대하여 차근차근 설명할 것이다.

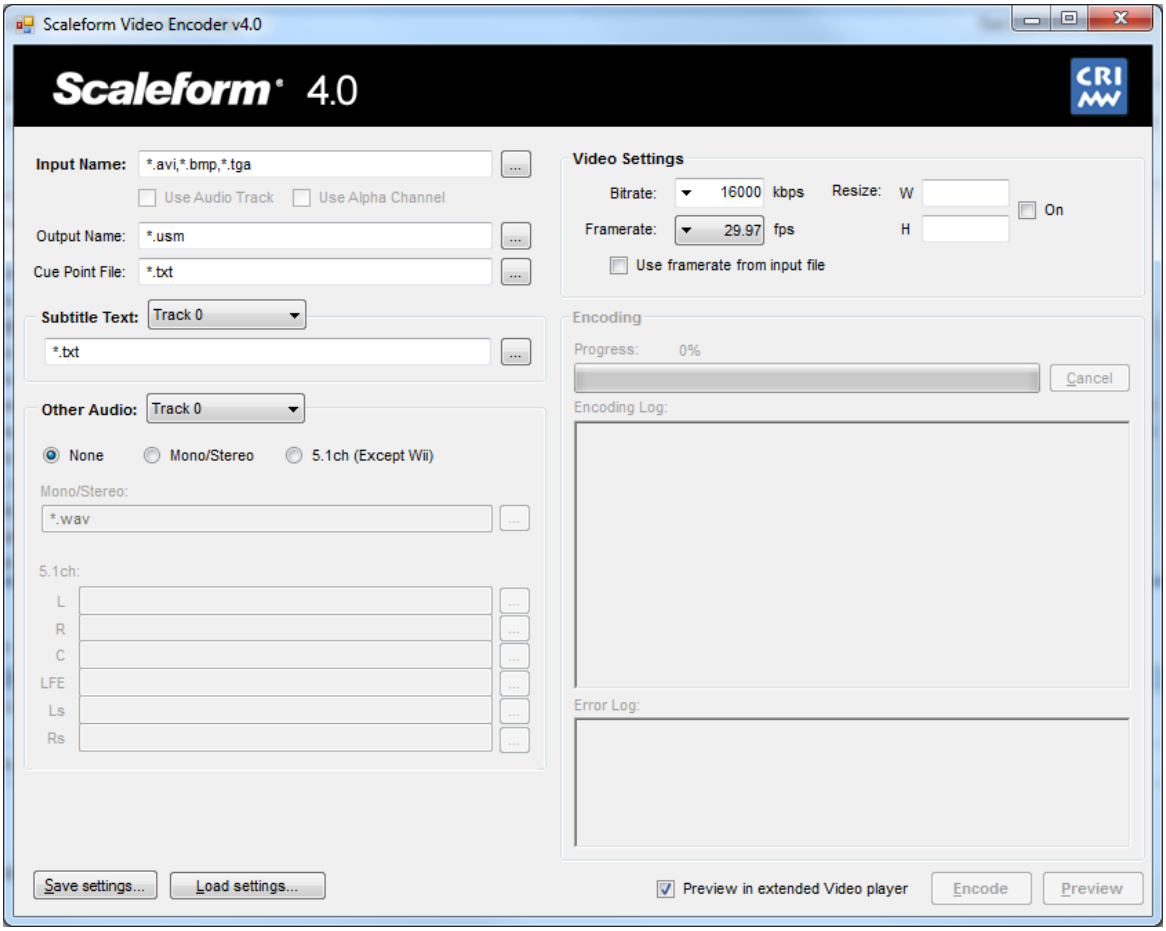

그림 2: 스케일폼 비디오 인코더

3.1 기본 인코딩-한걸음 한걸음

USM 포맷으로 변환하는 과정은 무지하게 쉽다. 자막이나, 큐포인트 등은 선택적으로 필요한 것이니까 논외로 하자. 여기서 우리는 알파채널(픽셀별 투명도), 큐포인트, 자막, 스테레오(2 채널)이 있는 USM 파일을 만드는 것에 대하여 보여줄 것이다.

윈도우 시작메뉴에서 스케일폼 인코더를 실행한다. 기본폴더에 설치했다면 다음 위치에 있을 것이다.

Windows Vista: *시작->모든 프로그램->Scaleform->GFx SDK 4.3->Video*

Windows XP: *시작->프로그램->Scaleform->GFx SDK 4.3->Video*

1. Browse 버튼[...]을 사용해서 변형할 AVI 포맷 비디오를 찾아 로드한다.
    a. 비디오는 AVI 나 연속하는 숫자로 지정된 BMP 나 TGA 파일도 가능하다
    b. 연속적인 숫자로 지정된 BMP 나 TGA 라면 첫 번째 파일을 선택한다.

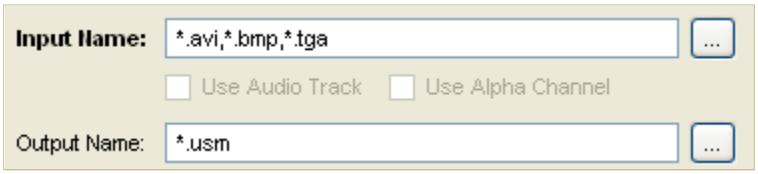

그림 3: 인코딩 될 비디오 파일은 Input Name 항목에 들어가야 함

2. 비디오에 오디오 트랙이 있다면 *Use Audio Track* 옆에 있는 체크마크에 체크 한다. AVI 에서는 기본값이 사용 가능하게 되어 있으나, BMP 나 TGA 에서는 사용불가 되어 있다.

3. 알파 채널이 있다면 *Use Alpha Channel* 옆에 있는 체크마크에 체크 한다. 이렇게 하면 알파채널이 완전 백색이 아니더라도 뷰어를 투과해서 그 밑에 있는 컨텐츠를 볼 수 있을 것이다. 기분적으로는 사용불가 되어 있다.

4. 소스 비디오 패쓰가 기본적으로 *Output Name* 에 채워져 있을 것이다. 하지만, *browse* 버튼을 사용해서 USM 파일을 다른 폴더에 저장할 수 있다.

5. *Encode* 버튼을 눌러서 인코딩을 시작한다.

3.2 최초 인코딩 예제

1. 인코더를 연다
2. *C:\Program Files\Scaleform\GFx SDK 4.3\Bin\Data\Tools\VideoEncoder\sample.avi* 를 로드한다.
3. 기본 설정 값으로 샘플무비를 인코드 한다.
4. 인코딩이 성공하면, Preview 버튼이 활성화 된다. USM 인코딩 비디오를 preview 로 본다.

3.3 큐 포인트

큐포인트는 챕터별 탐색, 데이터 교체, SWF 에서 이벤트를 발생시키는 등의 용도에 잘 쓰인다. 이번 장을 넘어가는 것도 가능하지만, 큐포인트에 관한 간단한 설명은 읽어두는 것이 좋을 것이다.

최종 인코딩 된 USM 에서 큐포인트를 사용하려면 Cue Point File 옆의 Browse 버튼을 사용해서 큐포인트 텍스트 파일을 위치시킨다.

그림 4: 큐 포인트 텍스트 파일 탐색

큐포인트 텍스트 파일에는 각 큐포인트 리스트가 줄 단위로 포함되어 있다. 파일의 제일 첫째 줄은 밀리초 단위의 출력 간격(의례적으로 1000)이다. 이 간격은 각 큐포인트의 시간 값을 계산하는 근거로 사용된다. 시간값 5000 은 아마도 5 초가 될 것이다(5000/1000=5). 각 줄의 간격 리스트 다음에 있는 각 큐포인트가 인코딩 될 때 사용된다.

2 가지 형태의 큐포인트가 있는데, 각각 네비게이션과 이벤트다. 네비게이션 큐포인트는 챕터 선택 시에 사용되는 것으로 DVD 메뉴의 챕터와 비슷하다. 이벤트 큐포인트는 인자값 설정에 사용되는 것으로 각 이벤트 큐포인트는 다양한 인자를 가질 수 있다. 인자들은 "키 = 값"형태로 나열된다. 다양한 인자들이 하나의 큐포인트에 나열될 수 있는데, 콤마(,)로 구분된다. 큐포인트를 플래시에 사용하는 방법에 대해서는 "액션스크립트에서 비디오 작업하기"를 참고하자.

한줄 큐포인트 포맷

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

사용예

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

예제 CutPoint.txt 파일의 내용

```
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 샘플 비디오를 큐포인트로 재인코딩 하기

1. 스케일폼 비디오 인코더 실행
2. *C:\Program Files\Scaleform\GFx SDK 4.3\Bin\Tools\VideoEncoder \sample.avi* 파일 열기
3. 텍스트 필드 옆의 Cue Point File 버튼을 눌러서 *sample_cue_points.txt* 파일 읽기(이 파일은 *sample.avi* 와 동일한 위치에 있음)
4. 기본 설정값으로 샘플무비를 인코딩
5. 인코딩이 완료되면 *Preview* 버튼이 활성화되므로 USM 파일을 프리뷰 해볼 것
6. 비디오 플레이어의 *fast forward* 와 *rewind* 버튼을 사용해서 큐포인트 간의 앞과 뒤로 이동해 볼 것

3.4 자막

자막 텍스트 파일은 각 자막이 줄단위로 나열되어 있다. 자막은 텍스트를 지정된 시간에 비디오에 출력하기 위해서 사용된다. 비디오에 자막이 없다면 이번 장을 건너뛰어도 된다. 하지만, 자막을 어떻게 추가하는 지를 이해해 두는 것이 나을 것이다.

다중 자막 파일을 지원(자막 트랙 당 하나의 파일)한다. 따라서 각 언어별 다른 자막 파일을 만드는 것이 가능하다. 언어에 따라서 자막의 길이가 다르기 때문에 시작점과 끝점이 다룰 수 있다.

1. Subtitle Text 옆에 있는 드롭다운 메뉴에서 자막트랙을 선택한다. 최대 32 개 까지 가능하다(트랙 0-31)

2. 트랙에서 사용한 자막 텍스트 파일을 browse 버튼으로 선택한다.

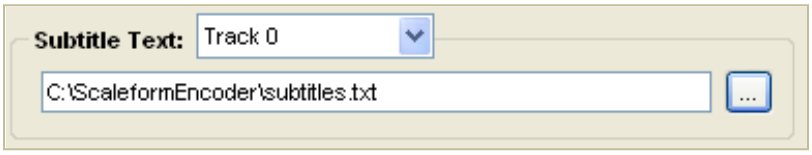

그림 5: 자막 트랙을 선택하고 파일을 고른다

자막 파일의 첫 번째 줄은(모든 주석 문은 제외) 출력 간격을 밀리 초로 환산한 값이다(일반적으로 1000). 이 값은 각 자막의 시작과 끝을 계산하는 근거 값으로 사용된다. 첫 줄 다음에 나오는 모든 줄은 3 가지 정보로 구성되어 있다. 시작 시간, 끝 시작, 메시지 ID 가 그것이다. 메시지 ID 는 텍스트 문자열을 출력하기 위한 코드로 사용 된다. 이때, 이 인자 대신에 실제 출력될 문자열이 사용되어도 된다. 상세한 설명은 "액션스크립트에서 비디오 작업하기"를 참고하라.

한줄 자막 포맷

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

예제 subtitle.txt 파일의 내용

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

3.4.1 자막으로 샘플 비디오 재인코딩

1. 스케일폼 인코더 열기

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

3. *Cue Point File* 옆의 browse 버튼으로 sample_cue_points.txt 파일 로드(이 파일은 *sample.avi* 와 같이 있음)

4. *Subtitle Text* 드롭다운 옆에 있는 browse 버튼으로 sample_subtitles.txt 파일 로드(이 파일은 *sample.avi* 와 함께 있음)

5. 기본 설정으로 인코딩 시작

6. 인코딩이 완료되면 바탕화면의 바로가기(Scaleform Media Player)를 이용해 Scaleform Player를 실행하고 *getting_started_with_video_tutorial.swf* 파일을 그 위로 드래그 앤 드랍한다.

7. 비디오 플레이어의 하단에 자막이 출력되는지 확인한다. 2-5 초 사이와 8-11 초 사이에 자막이 나와야 함

3.5 오디오

인코더의 오디오 채널은 다중 오디오 트랙을 모노/스테레오/5.1 채널 서라운드 형태로 추가할 수 있게 해준다. 각 채널은 다른 오디오 파일을 포함할 수 있기 때문에 다중언어 보컬 트랙을 인코딩 할 때 유용하다.

1. 인코더의 *Other Audio* 옆에 있는 오디오 트랙을 선택한다.

2. 오디오 타입 선택-none, mono/stereo, 5.1ch(Except Wii)

   주의: 이름에서 보는 것처럼 5.1 은 Wii 에서 쓸수 없다. 하지만 다른 플랫폼에서는 모두 지원된다.

3. Mono/Stereo 일 경우에는 Mono/Stereo 텍스트 항목의 Browse 버튼으로 WAV 포맷 오디오 소스를 선택

4. *Use Audio Track* 체크박스가 체크되어 있으면 이 필드에 이미 소스 비디오 파일의 패쓰가 설정됨.

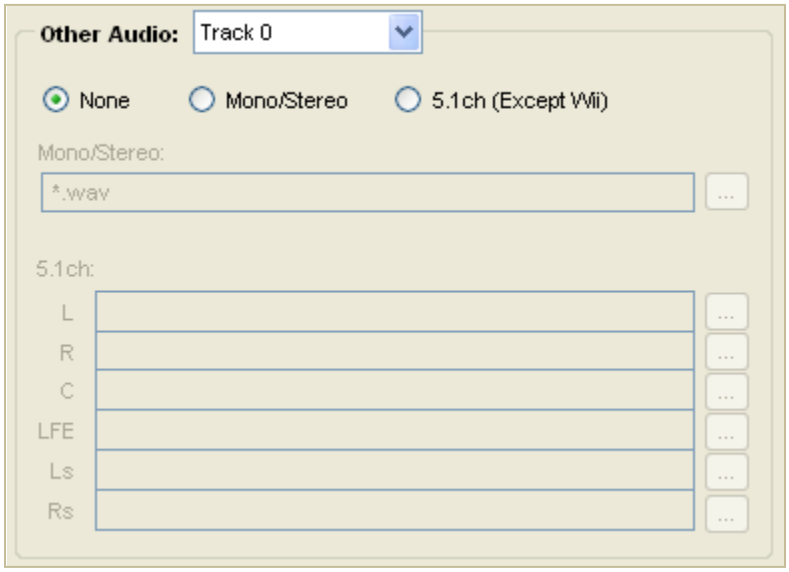

그림 6: 자막 트랙을 선택하고 파일을 고른다

5. 5.1ch 일 경우에는 각 항목의 Browse 버튼으로 각 채널(L,R,C,LFE,Ls,Rs)의 오디오 소스 선택

5.1 채널 서라운드 사운드의 채널:

L –좌측 스피커

R –우측 스피커

C –중앙 스피커, (음성으로 주로 사용)

LFE –서브 우퍼

Ls –후방 좌측 서라운드 스피커

Rs – 후방 우측 서라운드 스피커

3.5.1 두개의 트랙으로 예제 비디오를 재인코딩

1. 비디오 인코더 실행
2. *C:\Program Files\Scaleform\GFx SDK 4.3 \Bin\Tools\VideoEncoder\sample.avi* 로드
3. *Cue Point File 옆에 있는 browse 버튼으로 sample_cue_points.txt 파일 로드-이 파일은 sample.avi 와 동일 위치에 있음*
4. 항목 바로 밑에 있는 텍스트 항목 옆의 browse 버튼으로 sample_subtitles.txt 파일 로드- 이 파일은 sample.avi 와 동일 위치에 있음
5. *Other Audio* 항목을 Track 1 으로 설정
6. Mono/Stereo 라디오 버튼 선택
7. *sampe_audio.wav* 파일을 선택
8. 기본 설정 값으로 인코딩. 이때, 비디오 길이가 오디오 길이보다 길다는 경고가 발생할 수 있는데, 무시하면 된다.
9. 인코딩이 완료되면 활성화 되는 *Preview* 버튼으로 USM 비디오 프리뷰 재생
10. 우측 하단에 있는 *Audio Channel* 콤보 박스의 좌측, 우측 화살표 버튼을 눌러서 오디오 채널을 전환한다.

3.6 비디오 설정

비디오 설정은 필요한 퀄리티의 단계, 압축, 최종 비디오의 파일크기를 결정하는데 사용된다. (비트레이트, 프레임레이트, 폭, 높이 등)

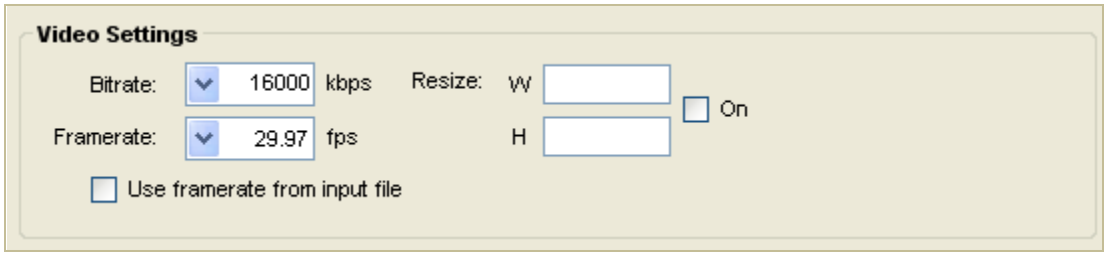

그림 7: 비디오 설정

1. 비트레이트 스텝퍼—비디오의 필요한 퀄리티 값 입력. 비트레이트는 초당 킬로바이트 수. 비디오에서 얼마만큼의 압축을 할 것인지 결정. 비트레이트가 클수록 퀄리티가 좋아지며, 파일 크기도 커진다. 비트레이트가 낮을수록 압축률이 높아지고, 퀄리티가 떨어지며, 파일 크기도 줄어든다.

   다음 표는 플랫폼 별 최적 인코딩 비율이다. 가끔은 낮은 해상도를 높은 비트레이트로 인코딩해서 첫 번째 인코딩 스피드를 올리는 경우가 있다. 그리고 나서 고해상도 비디오를 인코딩 하는 것이다.

플랫 폼별 추천 비트레이트(kbps)					
Video Format		**PS3**	**Xbox360**	**Wii**	**PC**
1080p	High Quality	40000	40000	n/a	40000
1080p	Standard	30000	30000	n/a	30000
1080p	High Compression	20000	20000	n/a	20000
720p	High Quality	36000	36000	n/a	36000
720p	Standard	26000	26000	n/a	26000
720p	High Compression	16000	16000	n/a	16000
480p	High Quality	16000	16000	8000	16000
480p	Standard	12000	12000	6000	12000
480p	High Compression	8000	8000	4000	8000

2. (옵션) 인코딩 되는 비디오의 프레임레이트를 맞춘다. 기본값은 29.97 fps(NTSC 의 표준값)이다. AVI 가 인코더에 로드 되면 *Use framerate from input file* 옆의 체크마크가 체크될 것이며, 인코더가 소스 비디오의 오리지널 프레임레이트를 사용하게 된다. 이 옵션의 체크를 없애고 프레임레이트를 임의 설정 할 수 있다. 프레임레이트란 1 초에 몇 장의 프레임을 보여줄 것인가를 결정하는 것이다. 표준 NTSC 텔레비젼 신호는 1 초에 29.97 프레임이므로 이 값이 권장되는 것이다. PC 모니터는 일반적으로 초당 60 프레임이거나 60 헤르츠다.
3. (옵션) 최종 출력 크기를 *Resize W* and *H* 항목에서 지정한다. 폭을 W 에 높이를 H 에 넣고 체크마크에 체크한다.
4. 인코딩하고, 인코딩이 완료되면 비디오의 프리뷰를 재생한다.
   비디오를 인코딩하면 USM 파일과 같은 디렉토리에 배치파일이 만들어질 것이다. 이 파일에는 명령행 인코딩에 필요한 명령들이 포함되어 있다.
5. 인코딩 과정이 끝나면 *Preview* 버튼으로 재생한다.
   Encode 버튼 옆의 *Preview in extended Video player option* 에 체크를 하지 않으면 재생컨트롤 없이 preview 가 가능하다.

4 시작하기: 비디오를 플래시에 추가

이번 장은 플래시 저작환경에 익숙하다고 가정할 것이며, USM 인코딩 된 파일이 있다고 생각할 것이다. 이번 장은 SWF 에 액션스크립트로 비디오를 추가하는 과정을 차근차근 보여줄 것이다. 이번 장은 기초설명이기 때문에 play, pause, rewind, 비디오 크기 조절 등의 각종 컨트롤을 추가하는 것은 생략할 예정이다. **이에 대한 상세한 방법은 NetStream 클래스에 관한 플래시 도움말을 참고하도록 하자.**

가장 먼저 할 일은 스케일폼 인코더로 USM 파일을 만드는 것이다. "시작하기: 스케일폼 비디오 인코딩"장을 참고하기 바란다. 이번 장에서 설명하는 내용은 USM 파일을 SWF 에 추가하고, 스케일폼 Scaleform Player 에서 SWF 파일을 재생하는 것이다.

4.1 플래시에서 비디오 설정

비어있는 AS 2.0 플래시 파일을 만들고 이 파일을 스케일폼 인코더가 있는 하드디스크 디렉토리에 저장한다. 간략한 처리를 위해서 USM 비디오 파일과 SWF 파일을 동일한 디렉토리에서 작업할 것이다.

1. 타임라인의 Layer 1 을 더블클릭해서 'mvplayer'으로 변경한다.

2. *New Layer* 버튼을 눌러서 타임라인에 새로운 레이어를 만들고 이름을 'actions'로 한다.

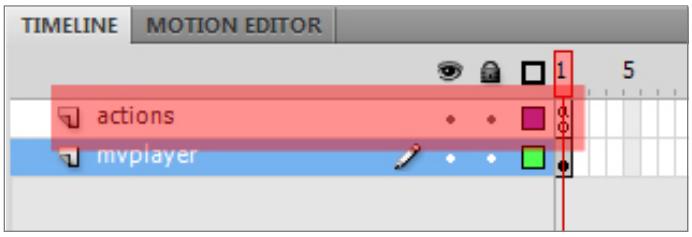

그림 8: 타임라인의 액션과 mvplayer 레이어

3. 타임라인 *actions* 레이어의 1 프레임을 선택하고 F9 를 눌러서 *Actions* 패널을 연다. 그리고나서 다음 코드를 입력한다.

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

주의: 만약 비디오 파일이 SWF 와 동일한 디렉토리에 있다면 USM 확장자를 포함한 비디오 파일명만 입력하면 된다. 하지만, 다른 곳에 파일이 저장되어 있다면 전체 패쓰명을 입력해야 하는데, 이때 백슬래시를 일반 슬래시로 해서 지정하기 바란다. - 예:
"*C:/pathtovideo/sample.usm*"

4. 타임라인의 mvplayer 를 선택

5. 스테이지에서 무비클립 생성
   a. 한가지 방법은 커다란 박스를 그리는데, 이 박스가 스테이지를 검은색 윤곽선과 투명 내부 채우기로 채우는 것이다.

b. 아무 검은색 윤곽선을 더블 클릭해서 전체박스 선택

c. 아무 검은색 외곽선에서 오른쪽 클릭하고 *Convert to Symbol* 선택

d. 무비클립 명을 'mvplayer'로 하고 OK 클릭

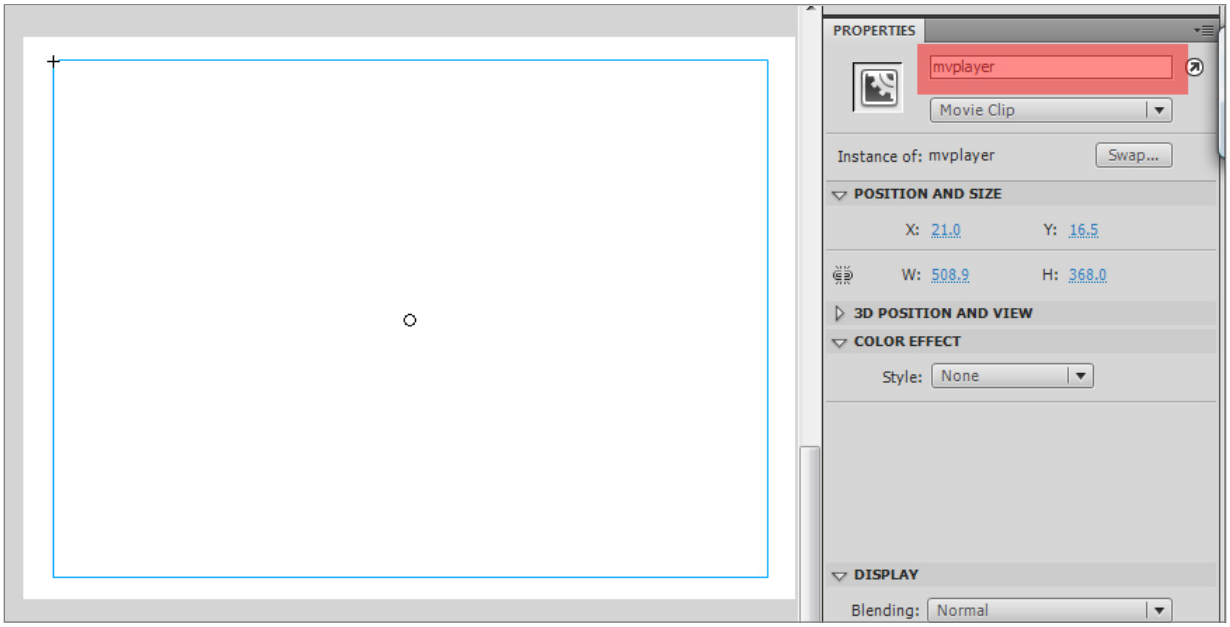

그림 9: 무비클립 만들기, mvplayer

6. 스테이지의 검은색 윤곽선을 한번 눌러서 mvplayer 무비클립 선택

7. *Properties* 탭에서 무비클립의 인스턴스 명을 'mvplayer'로 변경

8. mvplayer 무비클립을 더블클릭해서 타임라인을 들어간다.

9. *mvplayer* 무비클립내에서 새로운 레이어를 생성하고 이름을 'actions'로 정한다.

10. 타임라인 *actions* 레이어의 1 프레임을 선택하고 *Actions* 패널을 열어서 다음 코드 삽입

```
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);
}
```

주의: mvplayer 타임라인의 1 번 프레임 actions 레이어가 선택된 것을 확인하고, 코드를 입력하기 전에 *Layer 1* 에 그래픽 이미지가 없을 것

11. *Library* 창에서 우측 클릭해서 팝업메뉴의 *New Video* 선택

12. 'video'로 이름을 정하고 OK 클릭

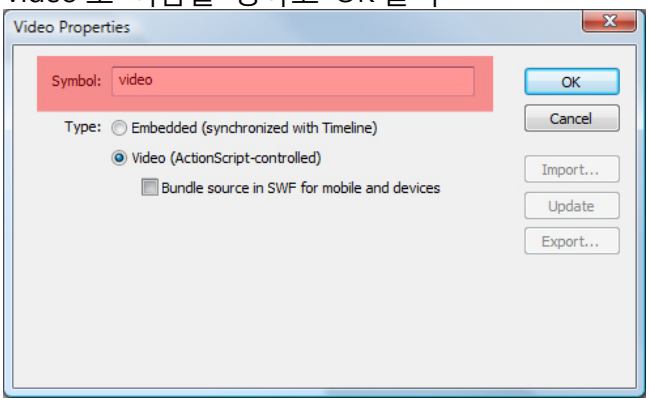

그림 10: 비디오 객체 만들기

13. *New Layer* 버튼을 눌러서 새로운 레이어를 생성. 이 레이어 명은 'Layer 3'일 것임

14. *Library* 창에서 new video 객체를 끌어다 *Stage* 의 *Layer3* 프레임 1 에 갖다 놓는다. 원하는 형태로 크기와 위치를 지정

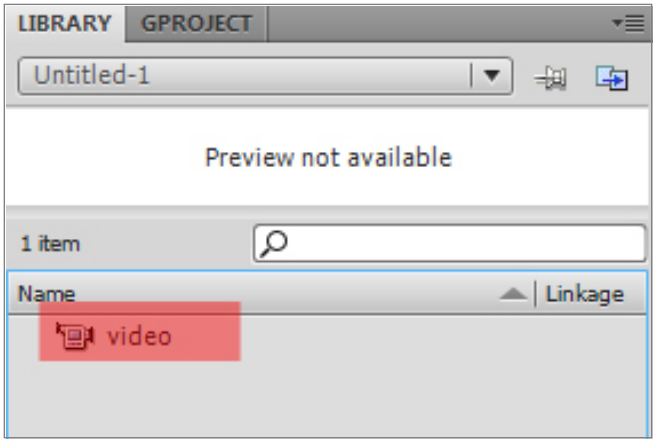

그림 11: 라이브러리 창의 비디오 객체

15. 스테이지에서 비디오 객체를 선택해서 *Properties* 탭의 인스턴스 명을 'video'로 변경

16. 저장하고 무비를 퍼블리싱 한다

4.2 비디오 테스트

일단 비디오가 들어간 SWF 파일이 생성되면 스케일폼 Scaleform Player 를 사용해서 비디오를 보는 것은 무지 쉽다.

1. 플래시의 스케일폼 런처 패널에서 SWF 를 실행

또는:

1. 스케일폼 Scaleform Player 열기

2. Scaleform Player 에 SWF 파일을 끌어놓기
    a. 혹은 SWF 파일을 Scaleform Player 파일 바로가기에 끌어놓기 해도 됨

5 시작하기: 액션스크립트에서 비디오 작업하기

완전히 인코딩 된 USM(자막, 큐포인트, 다중 언어 채널) 파일이 생성되고, 액션스크립트로 플래시파일에 비디오가 추가되면 자막, 오디오 채널 출력, 큐포인트 네비게이션, 큐포인트 이벤트 실행 등이 가능해 질 것이다. 주의: 이번 장에서는 "시작하기: 비디오를 플래시에 추가하기"에 기반해서 쓰여졌으므로 관련 플래시 파일을 먼저 만들어 두기 바란다.

이번 장에서는 SWF 파일의 큐포인트, 자막, 오디오트랙을 접근하는 방법에 대하여 설명할 것이다. 이 학습서는 액션스크립트에서 비디오 작업하는 것을 기반으로 쓰여졌다. 몇 가지 샘플 코드가 제공되겠지만 이 코드가 유일한 방법은 아님을 명심하자.

5.1 액션스크립트에서 큐포인트로 작업하기

USM 파일은 네비게이션이나 이벤트 포인트 등의 큐포인트와 함께 인코딩 될 수 있다. 네비게이션 포인트는 기본적인 챕터 이동에 유용하며, 이벤트 포인트는 키값인자를 플래시 파일에 전달해서 이벤트를 발생시키거나 속성값을 변경할 때 유용하다.

주의: 큐포인트 텍스트 파일로 인코딩된 USM 비디오를 재생하는 플래시 SWF 파일은 큐포인트에 접근할 수 있어야 한다. 이 문서를 충실하게 따라왔다면 독자는 이미 큐포인트를 포함해서 완벽하게 인코딩 된 USM 비디오를 가지고 있을 것이다. 만약 아직 준비가 안되어 있다면 "비디오를 플래시에 추가하기"나 "큐포인트를 비디오에 추가하기"장을 참고하기 바란다.

큐포인트에 접근하려면 액션스크립트에 몇 가지 추가할 것들이 있다. mvplayer 무비클립 *actions* 레이어의 1 번째 프레임에 다음 코드를 넣도록 하자. 액션스크립트 패널에서 직접 입력하거나 복사해서 붙여넣기 하면 된다.

1. *mvplayer* 무비클립을 더블클릭한다.
2. *mvplayer* 내의 타임라인의 actions 레이어에서 첫번째 프레임을 선택하고 다음 코드를 *Actions* 패널의 가장 끝에 넣는다.
3. 큐포인트 이벤트를 대기하다가 이벤트가 발생하면 traceCuePoint 함수를 실행한다.

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

4. traceCuePoint 함수는 큐포인트 객체를 인자로 받는다. 다음으로, 이 함수는 큐포인트 인자를 metaPropPJParams 객체에 보관한다. 그리고 나서, 함수는 큐포인트의 기반 정보를 추적(출력 윈도우에 출력)한다. 기반정보란 큐포인트 명(currentCuePoint.name),큐포인트 시각(currentCuePoint.time), 큐포인트 타입(currentCuePoint.type)이다. 다음에 소개하는 코드 대신에 큐포인트 데이터를 사용해서 비디오 재생(ns.time)을 하는 것도 가능하다.

```
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. 마지막으로 큐포인트는 인자를 갖는데, 다음에 소개하는 traceObject 함수를 사용하면 출력 창에 인자를 추적하는 것도 가능하다. 큐포인트 데이터는 SWF 에서 속성(변수)를 변경하거나 이벤트를 발생시키는데 사용할 수 있다.

```
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. SWF 파일을 Scaleform Player 파일에 끌어 놓기 하거나 런처패널을 사용해서 최종 플래시 파일을 테스트 한다. 모든 것이 완벽하다면 비디오의 큐포인트가 Scaleform Player 의 출력창에 보여야 한다.

5.2 액션스크립트에서 자막 작업하기

자막과 함께 인코딩된 비디오는 플래시에서 자막을 조작하거나 출력할 수 있다. 또한 다중 언어도 지원한다.

주의: 자막 있는 비디오를 사용하기 이전에 자막과 함께 인코딩 된 USM 파일을 재생하는 SWF 를 먼저 만들어야 한다. 이 문서를 잘 따라왔다면 이미 이런 파일이 있을 것이다. 아직 파일이 없다면 "비디오를 플래시에 추가하기"와 "자막을 비디오에 추가하기" 항목을 보기 바란다.

1. *mvplayer* 무비 클립 더블클릭

2. *mvplayer* 내에서 타임라인 상의 actions 레이어의 첫번째 키프레임 선택. *Actions* 패널에 있는 코드 제일 끝에 다음 코드 삽입

3. 일단, 자막 채널을 보관할 subtitleChannelNumber 변수를 생성해서 0 으로 셋팅한다

```
var subtitleChannelNumber = 0;
```

4. 다음으로, 비디오 파일의 메타데이터로부터 자막 채널의 개수를 얻는다. 주의: 자막채널 0 은 자막끄기다.

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

5. 스테이지에서 *mvplayer* 무비클립 내에 동적 텍스트 필드를 만들고 인스턴스 명을 'subtitle'이라고 정한다. 충분한 크기로 만들어서 눈에 띄는 컬러와 폰트를 선택하자.

6. 내장된 자막의 플레이 타임과 일치하면 다음 콜백 함수가 실행될 것이다. msg 문자열에는 USM 무비가 인코딩 될 때 입력된 자막 문자열이 들어가 있을 것이다. 변수명이

subtitleId2 인 두 번째 자막일 경우에는 변수명이 액션스크립트에서 메시지를 결정하는데 사용된다. Msg 의 내용은 5 번 단계에서 만든 *subtitle* 동적 텍스트 필드에 출력될 것이다.

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

7. SWF 파일을 Scaleform Player 파일에 끌어 놓기 하거나 런처패널을 사용해서 최종 플래시 파일을 테스트 한다. 모든 것이 완벽하다면 비디오의 자막이 출력창과 플래시의 텍스트 필드에 보일 것이다.

5.3 액션 스크립트에서 오디오 채널 작업하기

USM 은 다중 오디오 채널로 인코딩 될 수 있기 때문에 각각 필요한 언어로 분할된 채널의 비디오를 만들 수 있다. 이러한 오디오 채널을 SWF 에서 접근하는 것은 엄청 짧은 코드가 필요할 뿐이기 때문에 다루기 매우 쉽다.

오디오 채널 비디오를 사용하기 이전에 적어도 하나 이상의 오디오 채널로 인코딩된 USM 파일을 재생하는 SWF 를 먼저 만들어야 한다. 이 과정에 대한 간략한 해설은 "비디오를 플래시에 추가하기"와 "비디오에 오디오 채널 인코딩하기" 항목을 보기 바란다.
mvplayer 무비 클립의 *actions* 레이어의 1 번째 프레임에 다음 코드를 삽입한다.

1. *mvplayer* 무비클립을 더블클릭해서 타임라인으로 들어간다.

2. *mvplayer* 내에서 타임라인 상의 *actions* 레이어의 첫번째 키프레임 선택. Actions 패널에 있는 코드 제일 끝에 다음 코드 삽입

3. 2 개의 변수 선어. 첫번째 AudioTracks 는 비디오에 있는 각 오디오 트랙을 보관할 배열. 두번째 CurAudioTrack 은 현재 선택된 트랙 보관용. 이 변수는 0 으로 초기화.

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

4. 다음, 비디오의 메타데이터를 읽을 콜백함수 선언. 비디오의 audioTracks 메타데이터를 배열에 AudioTracks 대입. 주의: 만약 자막추가 콘트롤 장에서 추가한 'ns.onMetaData' 가 이미 존재한다면 trace("Subtitle Channels: " + subtitleChannelNumber); 과 블록의 끝을 나타내는'} '사이에 AudioTracks = infoObject.audioTracks;를 추가한다.

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

5. 테스트를 위해서 간단한 사각 그래픽 무비클립을 스테이지 상에 있는 *mvplayer* 무비 클립내의 아무 곳에 만들어 버튼으로 사용한다. 무비클립의 인스턴스 명은 'prev_audiotrack'이다.

6. *mvplayer* 의 actions 레이어의 첫번째 프레임의 *Actions* 패널에 다음 코드 삽입. 이 코드는 *prev_audiotrack* 버튼이 눌릴경우 오디오 채널을 후방으로 순환시켜준다.

```
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. 5 번째 단계에서 생성한 무비클립을 스테이지상의 *mvplayer* 내부에 사본으로 만들어서 'next_audiotrack'으로 인스턴스명을 변경한다.

8. *mvplayer* 의 actions 레이어의 첫 번째 프레임의 *Actions* 패널에 다음 코드 삽입. 이 코드는 next_audiotrack 버튼이 눌릴 경우 오디오 채널을 전방으로 순환시켜준다.

```
next_audiotrack.onRelease = function()
{
     if (AudioTracks  == undefined || AudioTracks.length < 2)
     {
           return;
     }
     CurAudioTrack++;
     if (CurAudioTrack >= AudioTracks.length)
     {
```

```
                CurAudioTrack = 0;
            }
            ns.audioTrack = AudioTracks[CurAudioTrack].trackIndex;
        }
```

9. SWF 파일을 Scaleform Player 에서 테스트한다. 모든 것이 완벽하다면 다양한 오디오 트랙을 *Next* 와 *Previous* 버튼으로 변경할 수 있을 것이다(물론 이것은 USM 파일에 다중 채널로 오디오 트랙이 들어가 있을 때 그렇다는 것이다).

6 액션 스크립트 비디오 확장기능

Scaleform 가 지원하는 플래시에서 비디오 파일을 재생시키기 위한 액션스크립트 메서드들이 있다. 추가적으로 Scaleform 에서 정의된 몇 가지의 새로운 확장기능이 있다. NetStream 내장함수의 속성, 메서드 및 이벤트에 대해서 더욱 많은 정보를 보려면 어도비 도움말을 찾아보도록 하라.

6.1 NetStream 속성의 내장함수 지원

`currentFps:Number`

표시 중인 초당 프레임 개수입니다.

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

`time:Number`

초 단위의 현재 재생중인 위치

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

6.2 NetStream 이벤트의 내장 함수 지원

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

USM 파일 재생 중, 삽입되었던 큐포인트에 도달했을 때 호출됨.

주의: 'onCuePoint'의 자세한 정보는 어도비 도움말을 참고.

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

플레이어에서 재생되는 USM 파일에 삽입된 설명 정보를 받으면 호출됨.

주의: 'onMetaData'의 자세한 정보는 어도비 도움말을 참고.

Scaleform Extensions

`audioTracks`

USM 파일 내부의 오디오 트랙 배열

예:

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

`subtitleTracksNumber`

USM 파일에 있는 자막 트랙의 총 수.

예:

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

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

NetStream 객체의 상태 변경 또는 오류에 대한 이벤트가 발생할 때마다 호출됩니다.

주의: 'onStatus'의 자세한 정보는 어도비 도움말을 참고.

6.3 NetStream 이벤트의 내장 함수 지원

`close()`

모든 데이터 스트림의 재생을 중지시키고, ns.time 속성을 0 으로 설정한다. 그 스트림을 다른 용도로 사용 할 수 있음.

Example: `ns.close();`

`pause([flag:Boolean])`

스트림의 재생을 일시 정지 또는 계속 재생함.

Example: `ns.pause(true);`

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

외부 비디오(USM) 파일의 재생을 시작함.

Example: `ns.play("myVideo.usm");`

`seek(offset:Number)`

스트림의 시작을 지정한 시간(초)에서 가장 가까운 키프래임을 찾음.

Example: `ns.seek(50);`

`setBufferTime(bufferTime:Number)`

버퍼링 될 영상 데이터의 양을 초 단위로 설정합니다. GFx::Video::Video 는 디스크에서 적당한 만큼의 데이터를 버퍼링하여 재생을 부드럽게 하고 디스크 읽기 시간을 줄입니다. 버퍼 크기는 해당 비디오의 비트레이트와 다른 비디오 파라미터에 따라 결정됩니다. 이 버퍼의 기본 크기는 1 초분의 재생을 유지할 수 있을 정도입니다.

예: `ns.setBufferTime(2.5);`

6.4 새로운 Scaleform 의 속성들

`subtitleTrack = Number`

NetStream 에 서 사용되는 현재 자막 트랙 넘버

Example: `ns.subtitleTrack = 1;`

`audioTrack = Number`

주 오디오 트랙은 NetStream 으로 사용됩니다. 비디오 파일에 영어나 스페인어 등 다중 오디오 트랙이 있을 경우, 특정한 언어에 맞춰 오디오 트랙을 설정하는 데 사용할 수 있습니다.

예: `ns.audioTrack = 2;`

`subAudioTrack = Number`

일반적으로 NetStream 에서 사용되는 보조 트랙 또는 SubAudio 트랙으로 영상과 함께 출력되는 대화 트랙이나 음향 효과를 재생합니다.

예: `ns.subAudioTrack = 3;`

`voiceTrack = Number`

5.1 채널 오디오의 센터 트랙(center track)을 교체합니다. 5.1 채널 음악의 음성 트랙을 변경할 때만 사용합니다.

예: `ns.voiceTrack = 1;`

`setReloadThresholdTime(reloadTime:Number)`

다시 불러오기 타이밍을 설정합니다. 디스크에서 비디오 데이터 버퍼를 가져오는 빈도를 결정합니다. 예를 들어 응용 프로그램이 버퍼 크기를 4 초(setBufferTime 사용)로 설정하고 다시 불러오기 한계점을 1 초로 설정했을 경우, GFx::Video::Video 는 초기에 해당 버퍼를 4 초분의 데이터로 채웁니다. 3 초분의 데이터가 디코딩 되어 소비되면, 남은 데이터 시간이 *다시 불러오기 한계점*보다 짧아지므로 GFx::Video::Video 가 버퍼를 다시 채우게 됩니다.

예: `ns.setReloadThresholdTime(1);`

`setNumberOfFramePools(numPools:Number)`

내부 비디오 버퍼 개수를 설정합니다. GFx::Video::Video 는 내부 메모리를 사용하여 디코딩된 프레임 출력하기 이전에 버퍼링합니다. 프레임 풀의 수가 많으면 높은 CPU 부하에서도 부드러운 재생이 가능해집니다. 초기값은 2 입니다.

예: `ns.setNumberOfFramePools(3);`

`setOpenTimeout(timeout:Number)`

open file timeout(USM 헤더 디코딩에 걸리는 시간)을 초 단위로 설정합니다. 초기값은 5 초입니다. 이 기능을 비활성화(timeout 을 무한으로 설정)하려면 0 을 입력합니다.

예: `ns.setOpenTimeout(5);`

```
currentFrame:Number
```

현재 재생 위치(프레임 단위)를 반환합니다.

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

```
Loop = Boolean
```

NetStream 의 loop 에 true 를 설정하면 false 가 입력될 때까지 반복 함.

Example: `ns.loop = true;`

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

주어진 사각 영역 값으로 비디오를 출력함.

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

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

USM 파일로부터 자막 문자열을 플레이어가 받았을 때 호출 됨.

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

7 비디오 생성에 대한 이해

비디오 편집이나 비디오 생성은 까다로운 과정일 수 있다. 특히, 비디오 편집이나 변환 경험이 없는 사람이라면 더 그럴 것이다. 비디오 편집은 상당히 많은 해상도와 설정 조합이 있을 수 있고, 그 결과에 따라서 재생시의 퀄리티나 성능에 상당한 영향을 미친다. 비디오 영상은 일반적으로 해상도, 화면비율, 비트 레이트, 프레임 레이트 등을 고려해서 부드러우면서도 좋은 화질로 재생되도록 한다. 모든 하드웨어 플랫폼과 게임엔진들은 최적 효율을 내기 위한 인코딩 방법이 미묘하게 다르다. 따라서, 몇 번 정도 인코딩에 대한 실험과 테스트가 필요할 것이다.

예를 들어서 전체화면에 단일 파일을 재생하는 것은 비교적 간단한 요구사항이 필요할 것이다. 왜냐하면 시스템 자원을 거의 독점할 수 있기 때문이다. 하지만, 여러 개의 무비를 스트리밍 하거나, 백그라운드 로딩을 하거나, 무비에서 알파를 사용하거나, 텍스처에 무비를 재생하거나, 혹은 다른 고급 기능을 사용한다면 다양한 설정을 실험해서 정확한 설정 값을 찾아야 할 것이다.

HD 1080p 비디오를 만드는 것은 XB360 과 PS3 용의 최고급 무비를 위한 최고의 접근 방법이며, 이를 위해서는 비디오 생성 전에 몇가지 요소들에 대한 테스트가 선행되어야 한다. 먼저, HD 비디오를 편집하는 것은 파일사이즈가 어마어마 하게 크기 때문에 살인적으로 느리고 어려울 것이다. 하이엔드 머신에서 비디오 편집이 이루어지는 것이 아니라면 HD 는 피하는 것이 좋다. HD 비디오, 특히 하이 비트 레이트인 1080p 비디오가 보기에는 멋질지 몰라도 재생 시에 엄청난 시스템 자원을 필요로 하며, 무식한 파일 크기가 될 것이기 때문에 어떤 해상도로 제작할지 빠르게 미리 결정해야 할 것이다.

1080p 비디오 영상을 제작할 때 고려해야 할 또 다른 것은 비디오 소스(라이브 액션, CG, 게임 영상 등) 파일이 무엇인가, 누가 작업하는가, 개발 시간과 예산은 얼마인가 하는 것이다.

720p 와 1080p 그리고 다른 HD 해상도를 실제로 비교해 보는 것은 중요하다. 가끔 더 작은 해상도에서도 충분히 괜찮게 보이면서도, 만들기 쉽고, 크기도 더 작으며, 편집도 간편하고, 스트리밍 하기도 좋은데, 실행 시 필요한 메모리도 적은 경우가 있다. 비디오를 만드는 것이 리얼한 영상을 만들어서 게임에서 어떻게 재생될 것인지에만 프로젝트의 목표로 두는 것이 아니라

반복적인 테스트를 통해서 파이프라인과 개발 프로세스가 어떻게 진행되는 지도 아는 것이 중요하다.

궁극적인 최상의 해상도로 영상을 만들어 놓고 다운스케일 하는 것이 최고의 방법이기는 하다. 스케일폼 비디오 인코더가 비디오 크기를 조절할 수 있기는 하지만 화질이 떨어지거나 영상이 왜곡되는 등의 문제가 있을 수 있다. 이는 특히 낮은 해상도의 영상을 업스케일 할 때 발생한다. 따라서, 고해상도로 제작한 영상을 다운스케일 하는 것이 한결 나을 것이다.

거의 대부분의 영상 편집 프로그램이 스케일폼 비디오와 결합 가능하다. 하지만 최적의 결과물을 만들려면 비압축 AVI 를 사용하는 것이 좋다. 스케일폼 비디오로 인코딩할 때는 특히나 필수사항이다. 다른 코덱(DIVX 등)을 사용해서 압축한 비디오를 스케일폼 비디오 포맷으로 인코딩하게 되면 압축에 대한 재압축이 되기 때문에 파일크기도 줄어들지 않을뿐더러, 화질에 있어서 매우 부정적인 결과를 끼친다.

비디오를 제작할 때 오디오 역시 중요한 고려대상이다. 모노, 스테레오, 5.1 채널 중에서 어떤 형태를 지원하는지 아는 것은 중요하다. 하지만 오디오에 대한 지역화가 훨씬 중요할 수도 있다. 음성 파일이 어떻게 만들어지고, 각 나라의 언어별로 번역되고 재녹음 되는 것인지 고려하는 것이 중요하다. 스크일폼 비디오는 다중 센터 채널 오디오 트랙을 사용할 수 있으므로 다른 언어에 대한 음성을 매우 쉽게 지원할 수 있다. 각 나라 언어별로 분리된 영상을 만드는 것보다, 하나의 영상에 다중 언어를 삽입하는 것이 나을 수 있다. 하지만 이렇게 하려면 음성 오디오가 비디오나 오디오 편집 프로그램에서 분리되어 저장되어 있어야 실행 시에 믹싱 될 수 있다.

다시 한번 말하지만, 비디오 제작 초기에 게임 비디오 영상의 모든 요구사항을 고민하고, 수행 테스트를 해보는 것이 개발 프로세스에서 쓸데 없는 시간 낭비를 줄여줄 것이다.

7.1 알파 채널

비디오 편집이나 컴퓨터 그래픽에 생소한 독자라면 알파채널이나 마스크라는 개념에 익숙하지 않을 수 있다. 스케일폼 비디오는 픽셀 별 알파채널이 가능하기 때문에 비디오 상의 각 픽셀은 불투명하거나 혹은 투명도 단계 값을 가질 수 있다. 일반적인 비디오 편집 프로그램에서 비디오는 완전히 투명하거나 부분적으로 투병할 수 있는데, 이는 투명도를 제어하는 채널에 입력되는 색깔에 따라 다르다.

일반적으로 TV 나 영화에서는 "그린 스크린"을 사용하는 것이 일반적인데, 이 경우에는 배우가 녹색(혹은 가끔은 청색) 스크린 배경에서 촬영되는 것이다. 이렇게 하면 배경을 지우기 쉽고, 배우를 다른 장면과 합성할 수 있다. 밝은 녹색이나 청색이 일반적으로 사용되는 이는 이들 색이 영화에서 거의 사용되지 않기 때문에 다른 영상과 충돌할 일이 없기 때문이다. 비디오는 완전 투명이나 부분 투명으로 인코딩 될 수 있다.

대부분의 프로그램에서 알파 채널을 추가하는 것은 비교적 쉽다. 일반적으로는 체크박스 버튼에 체크만 하면 된다. 하지만, 이렇게 알파채널이 있는 비디오는 렌더링 될 때 알파가 없는 것보다 2 배의 비용이 든다는 것을 명심하자. 실시간 게임엔진에서 사용할 때는 특히 주의해서 사용해야 할 것이다.

8 기술적 통합 지침

이번 장에서는 스케일폼 비디오를 게임엔진에 통합하는 방법에 대한 기술 정보를 제공해줄 것이다.

8.1 Scaleform 사운드 시스템 초기화

Scaleform 사운드 시스템을 초기화 하려면 GFx::Audio 클래스의 인스턴스가 GFx::Loader 객체에 셋팅되어야 한다. 이렇게 하는 목적은 SWF 스트리밍 사운드를 재생하기 위한 사운드 렌더러 오브젝트인 SoundRenderer 와 동기화 인자 제공하는 것이다. SoundRenderer 클래스는 추상 C++인터페이스라서 게임에서 구현되어야만 한다. Scaleform 는 기본적인 구현만 제공하는데 이는 FMOD 크로스 플랫폼에 기반해서 만들어져 있다. 이 구현으로부터 인스턴스를 얻으려면 Sound::SoundRendererFMOD::CreateSoundRenderer 를 호출하고 나서 FMOD::System 객체로 생성해야 한다.

사용예:

```
// Initializing the sound renderer object
   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<SoundRenderer> psoundRenderer =
               *SoundRendererFMOD::CreateSoundRenderer();
   if (!psoundRenderer->Initialize(pFMOD))
      exit(1);

   // Setting an instance of GFx::Audio on the loader
   Ptr<GFx::Audio> paudioState = *new GFx::Audio(psoundRenderer);
   loader.SetAudio (paudioState);
```

8.2 Scaleform 비디오 재생 시스템 초기화

Scaleform 비디오 재생 시스템을 초기화 하려면 Video 클래스의 인스턴스가 GFx::Loader 객체에 세팅되어야 한다. 이렇게 하는 목적은 어플리케이션이 비디오 파일을 재생하고자 할 때 비디오 플레이어의 인스턴스를 만드는데 있다. 비디오 파일에 오디오 데이터가 있으며 재생이 필요하다면 VideoSoundSystem 인터페이스의 인스턴스가 GFx::Video::Video 객체어 세팅 되어야 한다. Scaleform 는 SoundRenderer 인터페이스에 기반한 플랫폼 별 구현을 제공한다.

사용예:

```
 // Setting an instance of Video on the loader. An optional parameter
//  allows to specify the priority of the video decoding thread.
Ptr<GFx::Video::Video> pvc = *new Video(Thread::NormalPriority);
// Setting a video sound system instance which is based on SoundRenderer
//interface
if (psoundRenderer)
{
      pvc->SetSoundSystem(psoundRenderer);
}
// Setting a video sound system instance which is specific to a particular
//platform.
// Note: SetSoundSystem method should be called only once per an instance of
Video
 pvc->SetSoundSystem(Ptr<Video::VideoSoundSystem>(*new
                              VideoSoundSystemDX8(0)));

loader.SetVideo(pvc);
```

윈도우의 Video::Video 클래스가 VideoPC 로 명명되었고 3 개의 인자가 필요하다. 첫 번째 인자는 비디오 디코딩 스레드들의 순서를 정의한다. 두 번째 인자는 비디오 디코딩에 사용되는 스레드의 수를 정의하고, 세 번째는 각 디코딩 스레드를 위한 서로 잘 맞는(affinity) 배열이다.

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

사용예:

```
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);
```

PS3 와 XBox360 플랫폼을 위해서 Scaleform 는 Video 클래스의 특별버전을 제공하는데, 이들은 추가적인 초기와 인자가 필요하다.

PS3 는 GFx::Video::VideoPS3 라는 이름을 갖고 있으며 5 개의 인자가 필요하다.

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

spurs - CellSpurs 구조체의 포인터

spu_num - 비디오 디코딩에 사용될 SPU 개수.

ppu_num - 비디오 디코딩에 사용될 PPU 쓰레드 개수

ppu_priority - PPU 쓰레드의 우선순위

spurs_priorities- SPU 코어의 우선순위

사용예:

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

PPU 사용률 최소화 예제:

Scaleform 비디오는 초기화할 수 있으며, 이에 따라 PPU 사용률이 최소화됩니다. 예를 들어, 720p 비디오는 PPU 를 5% 미만으로 사용하면서 50~60FPS 의 재생 속도를 낼 수 있습니다. 이것은 메인 쓰레드가 비디오 디코딩 쓰레드보다 높은 우선권을 가져야 할 정도로 바쁠 때 발생합니다. 예를 들어서 게임 실행 중의 메인 쓰레드는 매우 바쁘게 돌아가므로 이러한 상황이 일어나게 됩니다.

디코딩 쓰레드의 PPU 사용률은 여타 쓰레드의 작업량에 따라 결정되는데, 예를 들어 메인 쓰레드가 바쁘지 않을 경우 디코딩 쓰레드는 PPU 의 약 40%를 사용합니다. 하지만, 아래와 같이 설정해두면, 메인 쓰레드가 바쁠 때 디코딩 쓰레드보다 높은 우선순위를 차지하게 되므로, 디코딩 쓰레드는 자원의 4% 또는 5% 정도만을 사용하게 됩니다. 실제로 우선순위만 올바르게 설정되어 있으면 디코딩 쓰레드의 PPU 사용률은 1% 정도에 그칠 수 있습니다.

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

XBox360 은 VideoXbox360 라는 이름을 가지며 비디오 디코딩 시에 사용할 프로세서 코어를 지정하는 하나의 인자만 필요하다. 첫 번째 인자는 내보적인 동작을 담당하는 2 개의 스레드를 정의하여 이 스레드들이 가능한 한 중복되지 않도록 한다. 두 번째 인자는 비디오 디코딩 스레드들의 순서를 정의한다. 세번째 인자는 비디오 리더를 위한 추가적인 하드웨어 스레드를 정의한다.

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

사용예:

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

8.3 백그라운드 게임 데이터 로딩 API

게임 데이터 의 백그라운드 로딩은 비디오의 재생장치가 Video::VideoBase::ReadCallBack 이라는 콜백 인터페이스와 VideoBase::IsIORequired, VideoBase::EnableIO 메서드로 구현되었다.

ReadCallback 은 비디오 작업이 요청될 때와 그것이 완료되었을 때 어플리케이션에게 통보하기 위한 콜백 인터페이스이다.

예:

```
// 비디오 시스템을 위한 간단한 읽기 콜백 구현
class VideoReadCallback : public VideoBase::ReadCallback
{
   public:
   VideoReadCallback() {}
   virtual ~VideoReadCallback() {}

   virtual void OnReadRequested()
   {
      // 비디오 읽기 시작
      VideoReadStart = Timer::GetTicks();
      // ...
   }
   virtual void OnReadCompleted()
   {
      // 비디오 읽기 완료
      GameReadStart = Timer::GetTicks();
      // ...
   }

   UInt64  VideoReadStart;
   UInt64  GameReadStart;
};

// 비디오 시스템 초기화
Ptr<Video::Video> pvc = *new Video::VideoPC(
         Thread::NormalPriority, MAX_VIDEO_DECODING_THREADS, affnts);

// 비디오 읽기 콜백의 생성 및 설정
Ptr<VideoReadCallback> pvideoReadCallback = *new VideoReadCallback;
pvc->SetReadCallback(pvideoReadCallback);
```

IsIORequired 는 비디오 데이터에 대한 읽기 작업에 대한 수행이 비디오 시스템에 필요한지의 여부를 결정한다. 이 메서드는 true 를 리턴 하자마자 어플리케이션은 즉시 모든 디스크의 IO 작업을 멈추고 *EnableIO* 의 호출로 비디오 읽기 작업이 가능하도록 한다.

EnableIO 는 게임 데이터의 백그라운드 로딩을 위해 비디오 읽기 작업을 가능/불능 상태로 만든다.

예:

```
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");
}
```

비디오의 백그라운드 로딩의 기능은 두 가지의 풀 소스 코드 데모(VideoBGLoadFlash 와 VideoBGLoadData)로 제공되며 *Apps\Samples\Video* 경로에 있다.

- VideoBGLoadFlash – 이 데모는 비디오가 재생되는 중에 플래시 콘텐츠 로딩에 대한 간단한 구현이다.
- VideoBGLoadData – 이 데모는 어플리케이션에서 비디오가 재생되는 중에 어떻게 임의의 데이터를 로드 할 수 있는지에 대한 예제를 제공한다.

8.4 Scaleform VideoSoundSystem 인터페이스

GFx::Video::VideoSoundSystem 은 Video::Video 를 재생하는데 사용되는 사운드를 지원하는 추상 인터페이스이다. 개발자들은 이 클래스를 대치함으로써 비디오 사운드를 구현할 수 있다. 이 클래스의 인스턴스는 비디오를 재생하기 이전에 Video::SetSoundSystem()의 설치 및 생성이 필요하다. 일반적인 특정 플랫폼에 있어서 이 인터페이스를 다르게 구현함으로써 사용 될 수 있다.

사운드 시스템 인터페이스들은 Video 에 포함되어 있다.

- Video::VideoSoundSystemDX8 – Windows 를 위한 DirectSound
- Video::VideoSoundSystemXA2 – Windows 와 Xbox360 를 위한 XAudio2
- Video::VideoSoundSystemPS3 – PS3 를 위한 MultiStream
- Video::VideoSoundSystemWii – Wii 시스템 사운드
- Video::VideoSoundSystemFMOD - FMOD 기반의 사운드 인터페이스
- Video::VideoSoundSystemWwise – Wwise 기반의 사운드 인터페이스

현재의 Scaleform 버전에서 비디오 사운드에 대한 지원은 내장 플래시 사운드 재생 기능으로부터 분리되어있고, 일반적인 사운드 엔진을 필요로 하지 않는다. 이 기능을 위해, 비디오는 Scaleform 의 외부에서 사용되는 Sound::SoundRenderer 로부터 분리된 독립적인 Video::VideoSoundSystem 클래스로 지원된다. (자세한 사항은 8.1 절을 참고) 이것은 비디오 사운드 지원에 있어서 VideoSoundSystem 과 VideoSound 클래스만 구현 하면 되고, 이것은 SoundRenderer 보다 대단히 간단하다는 것을 의미한다. 만약 SoundRenderer 의 구현이 이미 되어있다면, 그것은 상위의 기능을 제공하므로 직접 Video 를 초기화하여 그것을 사용할 수 있음에 유의하라. 또한 어떤 경우에는 두 가지의 구현을 섞어서 사용할 수도 있다. (사용자 정의 사운드 클래스가 일반 사운드 엔진 보다 스트리밍 지원을 제공하는 경우에 도움이 됨.)

예:

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

비디오 사운드를 지원하기 위해서는 GFx::Video::VideoSoundSystem 와 GFx::Video::VideoSound 는 구현되어야 한다. 일반적으로 비디오의 초기화 과정에 VideoSoundSystem 의 인스턴스는 하나만 존재한다. VideoSoundSystem 은 독립적인 비디오 사운드 시스템을 나타내는 VideoSound 객체를 만드는데 사용되는 하나의 메서드, 즉 Create 를 노출한다. Scaleform 는 새로운 비디오가 열릴 때마다 이 함수를 호출할 것이다. (여러 비디오를 동시에 재생할 수 있음.) 각각의 VideoSound 객체가 생성된 이후에, Scaleform 는 오디오 출력을 시작하고 멈추기 위해 그것의 다양한 함수들을

호출할 것이다. 스트림을 위한 실제의 사운드는 전달된 VideoSound::PCMStream 폴링을 통해 스트림을 가져온다.폴링은 일반적으로 VideoSoundSystem 은 활성 사운드 서비스에 의해 유지 관리되는 별도의 스레드에 의해 이루어진다.

현재의 VideoSoundSystemWwise 는 Wwise SDK 의 v2009.2.1 build 3271 버전을 기반으로 구현되었으므로 그 이후의 버전을 사용할 수 있음에 주의하라. Audiokinetic 은 전체 소스 코드와 이 플러그인의 비주얼 스튜디오 솔루션/프로젝트 파일을 제공하고 있고, 이것들을 S*DK\samples\Plugins\AkAudioInput* 에서 찾아볼 수 있다. 자세한 사항에 대해서는 Wwise 의 문서를 참고하라. GFx::Video::Video 에는 Wwise SDK 의 어떤 것도 포함되어있지 않으므로 따로 설치를 해야만 한다.

예:

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

Apps\Samples\FxPlayer and *Apps\Samples\Common* 경로에 있는 Scaleform Player – FxPlayerAppBase.cpp, FxSoundFMOD.cpp, FxSoundWwise.cpp 등의 소스코드를 참고하고, 사운드 시스템 인터페이스들의 구형에 있어서의 자세한 사항은 *Src\Video* 의 경로에 있는 Video_VideoSoundSystem*.cpp 를 참고하라.

8.5 다중 언어 비디오

표준 다중 언어 비디오에는 각 언어별로 멀티 오디오 트랙이 있습니다. GFx::Video::Video 는 ActionScript 비디오 익스텐션을 사용하여 효과적으로 다중 언어 영상 데이터를 처리할 수 있도록 합니다. 영어와 스페인어로 된 스테레오 오디오가 적용된 비디오를 예로 들어봅시다. 이러한 비디오를 재생할 때 각 언어에 맞추어 오디오 트랙을 선택할 수 있습니다. 오디오 트랙을 선택하려면 *NetStream* 에 포함된 audioTrack 익스텐션을 사용합니다.

예: `ns.audioTrack = 2;`

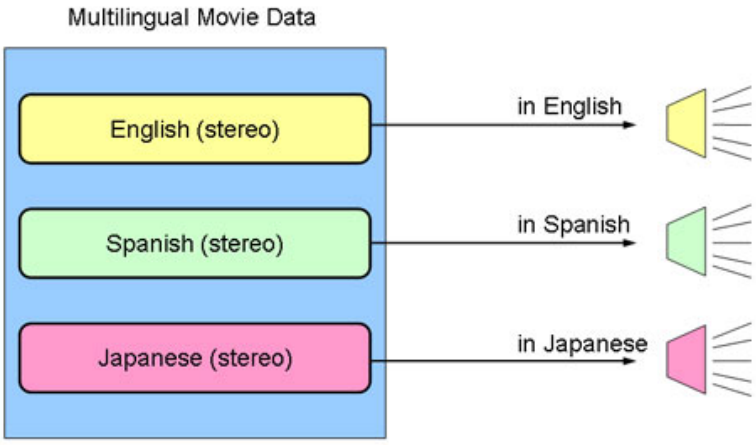

그림 12: 다중 언어 비디오

다중 언어 입체 음향 비디오는 일반적인 5.1 채널 오디오 트랙과 다중 언어 트랙을 포함합니다. 언어 트랙은 입체 음향에 따라서 변경될 수 있지만, 오직 하나의 서라운드 트랙만 다중 언어 트랙과 함께 사용 가능합니다. 이처럼 다중 서라운드 비디오에서 하나의 사운드 트랙을 공유할 경우 영상 크기가 줄어듭니다.

GFx::Video::Video 는 다음과 같이 두 가지 재생 기능을 지원합니다.

1. 중앙 채널 교체(Center Channel Replacement)
2. SubAudio 재생

8.5.1 중앙 채널 교체(Center Channel Replacement)

앞서 언급한 것처럼 다중 언어 입체 음향 비디오는 영상 파일 크기를 줄여 다양한 언어를 지원합니다. 다중 언어 서라운드 비디오는 일반적인 5.1 채널 오디오 트랙과 각 언어에 대응하는 모노럴(monaural) 음성 트랙을 포함합니다. 일반적으로 모노럴(monaural) 음성 트랙은 5.1 채널 오디오의 원본 센터 트랙(center track)을 대신하여 재생됩니다. 다중 언어 입체 음향 비디오를 재생할 때는 5.1 채널 오디오 트랙과 모노럴(monaural) 음성 트랙을 개별적으로 선택해야 합니다. 음악이나 음향 효과를 중앙 채널에서 재생하려면 이들을 미리 음성 트랙과 혼합해 두어야 합니다.

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

주 트랙이 5.1 채널 오디오가 아니거나 음성 트랙이 모노럴(monaural)이 아니면 음성 트랙 설정은 무시됩니다. 동일한 주 오디오 트랙이 음성 트랙으로 설정되어 있으면 중앙 채널이 정상적으로 재생됩니다.

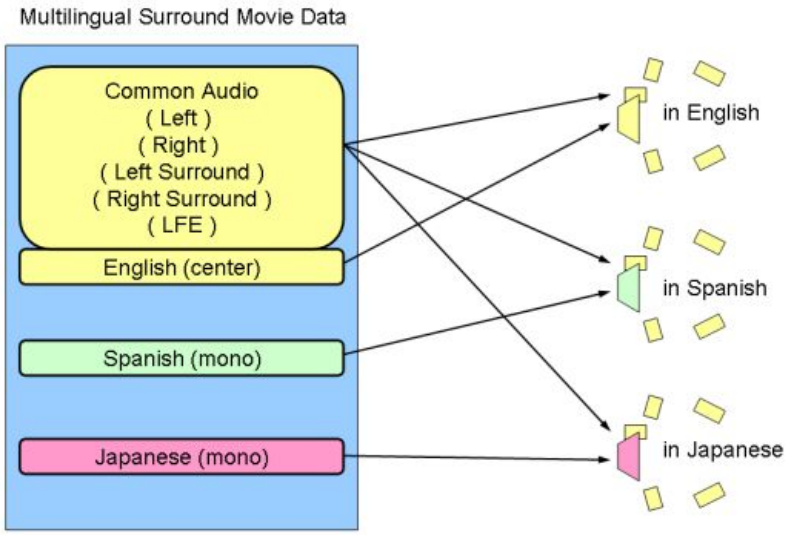

그림 13: 중앙 채널 교체(Center Channel Replacement)

8.5.2 SubAudio 재생

SubAudio 재생 기능을 사용하면 응용 프로그램이 GFx::Video::Video 를 통해서 주 트랙과 다른 트랙을 동시에 재생할 수 있습니다. 정확히 말하면, GFx::Video::Video 는 주 트랙뿐만 아니라 추가 사운드 인터페이스를 통해서도 사운드 데이터를 출력합니다. 따라서, 응용 프로그램은 SubAudio 트랙을 음성 트랙으로 출력하여 다중 언어 입체 음향 재생을 수행할 수 있습니다.

예:

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

SubAudio 재생 기능은 주 오디오 트랙의 볼륨 제어와 독립적으로 동작한다는 것에 주의해야합니다. 이것은 확장된 구문인 Sound.setVolume() 를 사용함으로써 제어할 수 있습니다.

예:

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

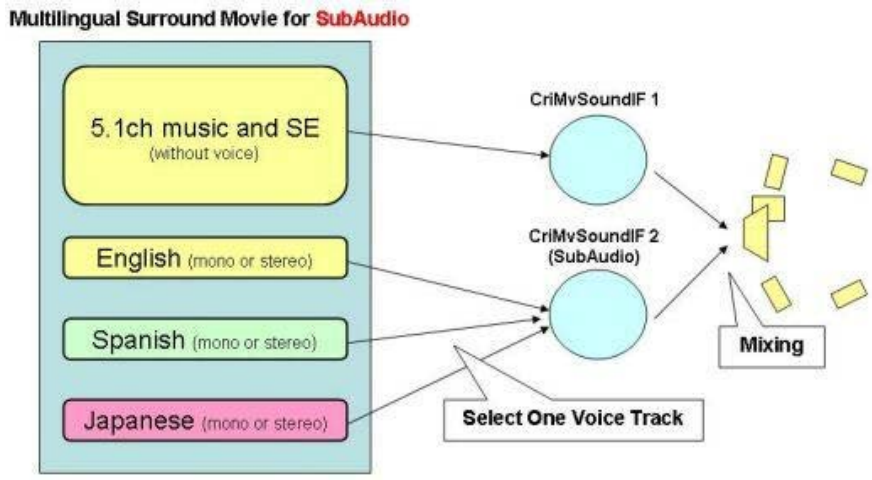

그림 14: SubAudio 재생

9 문제 해결

비디오 콘텐츠가 부드럽게 재생되지 않고 멈칫거림을 경험한다면 일반적으로 하드웨어에 비디오 데이터를 처리할 수 있는 리소스가 충분히 없다는 것을 나타냅니다. 종종 다음 방법에 의해 해결될 수 있습니다.

1. 코드에서 리소스를 해방합니다(즉, 비디오가 재생될 때 동시에 무거운 처리나 IO 부하 발생이 없도록 확인).

2. 또는 비디오 데이터의 크기와 비트레이트를 줄입니다.

3. 또는 비디오 시스템에 리소스(쓰레드, 프레임 풀)를 더 제공합니다

특히, 부드러운 비디오 재생을 돕기 위해 조정할 수 있는 세 가지 방법이 있습니다(우선 순위 순).

1. *쓰레드 구성* – 번호와 우선 순위를 변경합니다. 어떤 쓰레드가 이미 게임 엔진에서 사용 중이고 어떤 쓰레드가 사용 중이 아닌지를 고려합니다. 고해상도 비디오(720P 이상)에는 일반적으로 3 개 또는 4 개의 쓰레드가 필요합니다. 쓰레드 구성에 대한 자세한 내용은 위 섹션 8 을 참조하십시오.

2. *비디오의 해상도 및 비트레이트* –작업 부하를 낮추기 위해 비디오 동영상의 크기 및/또는 비트레이트를 줄여야 할 수 있습니다. Scaleform 은 필요에 따라 뷰포트에 맞게 자동으로 낮은 해상도의 비디오의 크기를 확대합니다.

3. *버퍼 및 프레임 풀* – 비디오 재생 구성을 위해 사용할 수 있는 많은 ActionScript 메소드와 익스텐션이 있습니다. 아래의 호출 두 개는 메모리 및 비디오 시스템에서 사용되는 버퍼링 시간 조정에 사용될 수 있습니다. 자세한 내용은 위의 섹션 6 을 참조하십시오.
   a. `NetStream.setNumberOfFramePools(numPools:Number)`
   b. `NetStream.setBufferTime(bufferTime:Number)`