

Adputer Composer

Version 3.6.0

©2009 ADPUTER INC.

Table of Contents

[Chapter 1. Introduction](#)

Technologies

Features

[Chapter 2. Architecture](#)

Diagram

Flat GUI

Customization

Keyboard Control

[Chapter 3. Technologies](#)

High-Definition Video

Multi Channel Video Mixing

HLSL Shader

FX Effects

Effects in Adputer Composer

.....

[Chapter 4. Applications](#)

Digital Signage

Surveillance

VJ Control

Video Communication

Video Remix

Mobile Video

[Appendix](#)

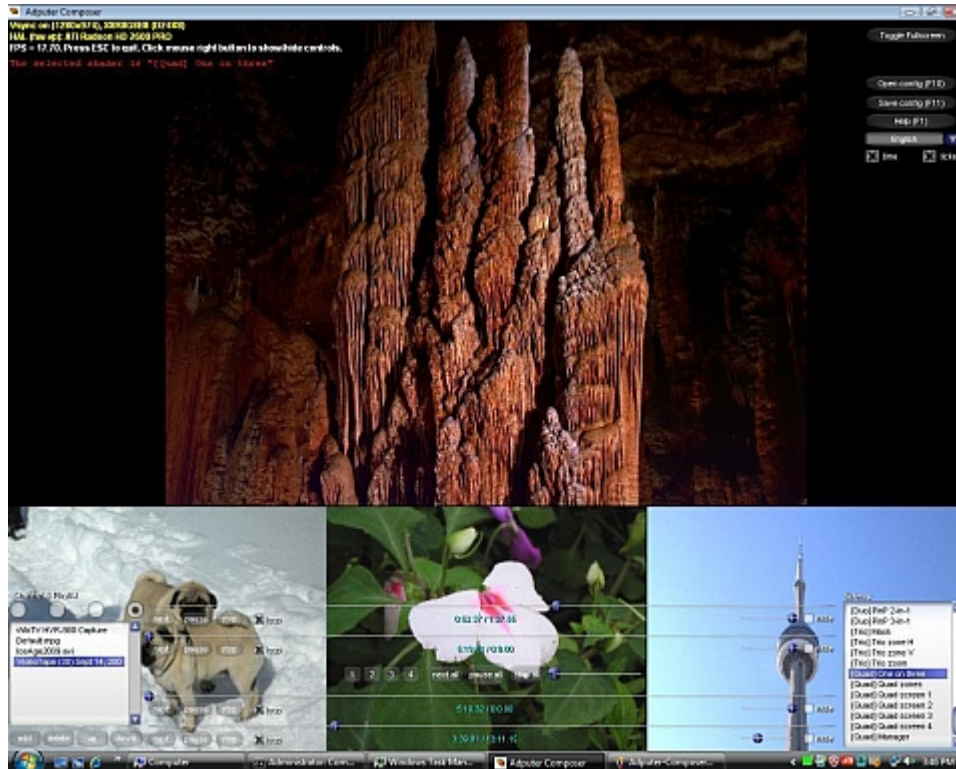
FAQ

History

System Requirements

Chapter 1 Introduction

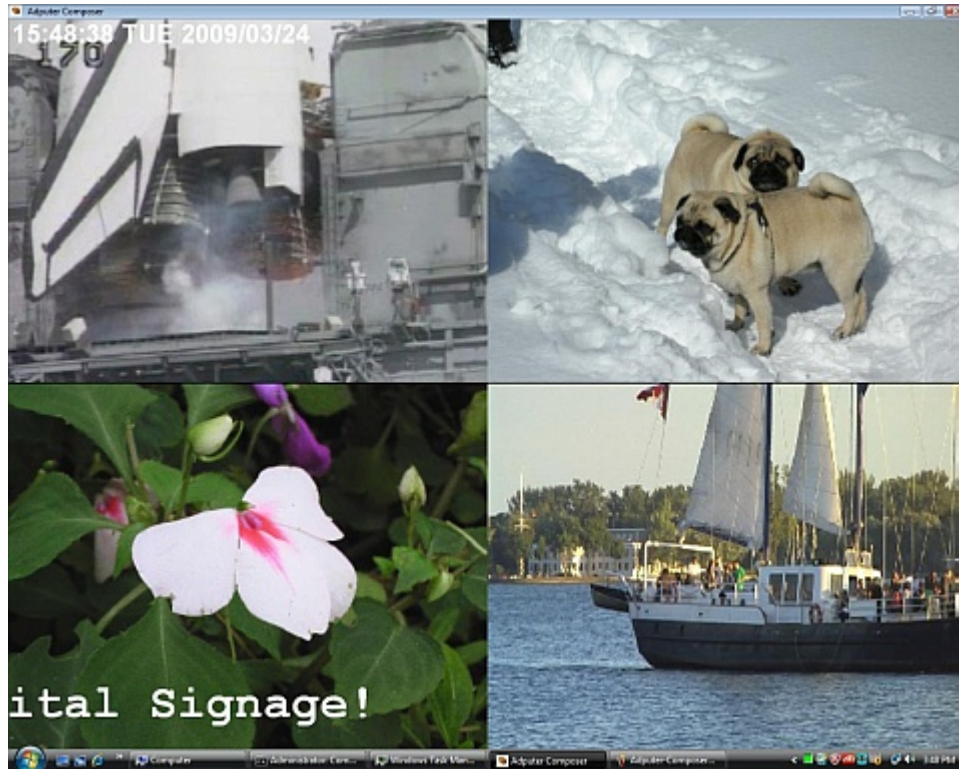
Adputer Composer is a parallel multi-channel video mixer with effects for Full HD and broadcasting video applications. By using the high-performance of multi-core CPU and hundred stream core GPU, Adputer Composer can finish video processing in real-time. Adputer Composer is greatly suitable for applications such as *Digital Signage*, *Video Surveillance*, and *Video Broadcasting*.



Adputer Composer applied *real-time mChannel* technology, which includes multi-thread media engine naturally for multi-core CPU (Central Processing Unit), programmable HLSL (*High Level Shading Language*) shader for GPU (Graphics Processing Unit), and media codec for *MP4 HD*, *WMV HD*, and *AVI* videos. Adputer Composer also supports mobile media applications along with wireless communication.

Technologies

mChannel technology allows multiple video streams to flow into a multi-thread pipeline for each channel. Each pipeline applied video splitter and decoders to split video and audio stream and further uncompress the media stream. Each uncompressed video streams are rendered and mixed in the GPU through a programmable shader. The final result is sent to frame buffer of the graphics card.



In addition to the **mChannel** technology, the design of **flat, transparent, and instant GUI** in Aputer Composer makes the use of player much easy. The technology of embedded SQL database management also makes play list and customized playback feasible.

Key Features

- Cutting-edge technologies and compact design
- Small, fast, powerful multi-thread video engine
- Real-time FX effects as chroma key and wipe on GPU
- Rich programmable shader with HLSL language
- Multi-channel video streams for video composition
- Multiple zone for independent video display
- Flat GUI for instant channel control
- Real-time ticker display for text message
- Customized player supports multiple windows and screens
- International languages support
- Recognize many media formats: MP4, WMV, AVI, JPEG, MP3
- Embedded SQL database for configuration management
- Support TV tuner for cable TV
- Support DV camera, Web camera

1. Cutting-edge technologies and compact design

Cutting-edge technologies inside Adputer Composer includes multi-thread video engine, multiple channel video mixing, and programmable special visual effects by GPU. Because of the streaming and pipeline architecture, a GPU can be up to 20x faster than a CPU in video processing.

2. Small, fast, powerful multi-thread video engine

Adputer Composer can speed up the execution of multi-thread video engine through multi-core processors. Since each channel may need to create many threads, Adputer Composer is naturally benefit from processors of quad core or more.

3. Real-time FX effects as chroma key and wipe on GPU

Quad video streams can be mixed and processed by an HLSL shader on GPU in real-time. You can switch to a new shader instantly by moving an arrow key or click mouse button by selecting an item in the shader list.

4. Rich programmable shader with HLSL language

Advanced GPU makes programmable shader available. Adputer Composer supports real-time video shader via FX effects in Windows XP and Vista. *Shader Model 4.0/3.0/2.0x/2.0*, High Level Shader Language (*HLSL*), Instant Compiler, FX Effects are supported in the Adputer Composer along with DirectX9.0 and plus.

5. Multi-channel video streams for video composition and post-processing

Adputer Composer can receive and process up to eight video streams. All of them can be composited on GPU in real-time with special effects. New multiple GPU technology such as SLI and Crossfire can allow more and high resolution video streams to be rendered simultaneously.

6. Multiple zone for independent video display

In Adputer Composer a new technology has been applied to display multiple video streams in zoning. So Adputer Composer can display four video streams on quad sub-windows. This function helps video contend makers to composite and manage videos easily for the applications such as digital signage and video communication.

7. Flat GUI for instant channel control and ticker

All the controls of media playback and settings are showed over the video surface. Users can select shader, play or pause videos, and adjust the volume of sound on each channel. The substitution buttons allow users to exchange channels for active processing.

8. Customized player supports multiple windows and screens

User can use sqlite database configure file to launch multiple instances of windows or display them in multiple screens.

9. International languages support

International languages are supported to help various users in the world. The supported languages include English, French, Chinese, Japanese, Korea, Dutch, Germany, Italian, Spanish, and Portuguese.

10. Recognize many media formats: WMV, AVI, MPEG, JPEG and HD videos

Adputer Composer supports codec that is compatible with DirectShow in Windows. It also supports super high-resolution images up to 8192x8192 (8K) pixels in DirectX 10 capable graphics card.

11. Recognize many media formats: MP4, AVI, JPEG, MP3 and HD videos

Most popular media formats are supported.

12. Embedded SQL database for configuration management

The configuration file of Adputer Composer is stored in the .sqlite database. Users can apply Sqlite database manager and command line to control the startup and resume playback of Adputer Composer.

13. Support TV tuner for cable TV

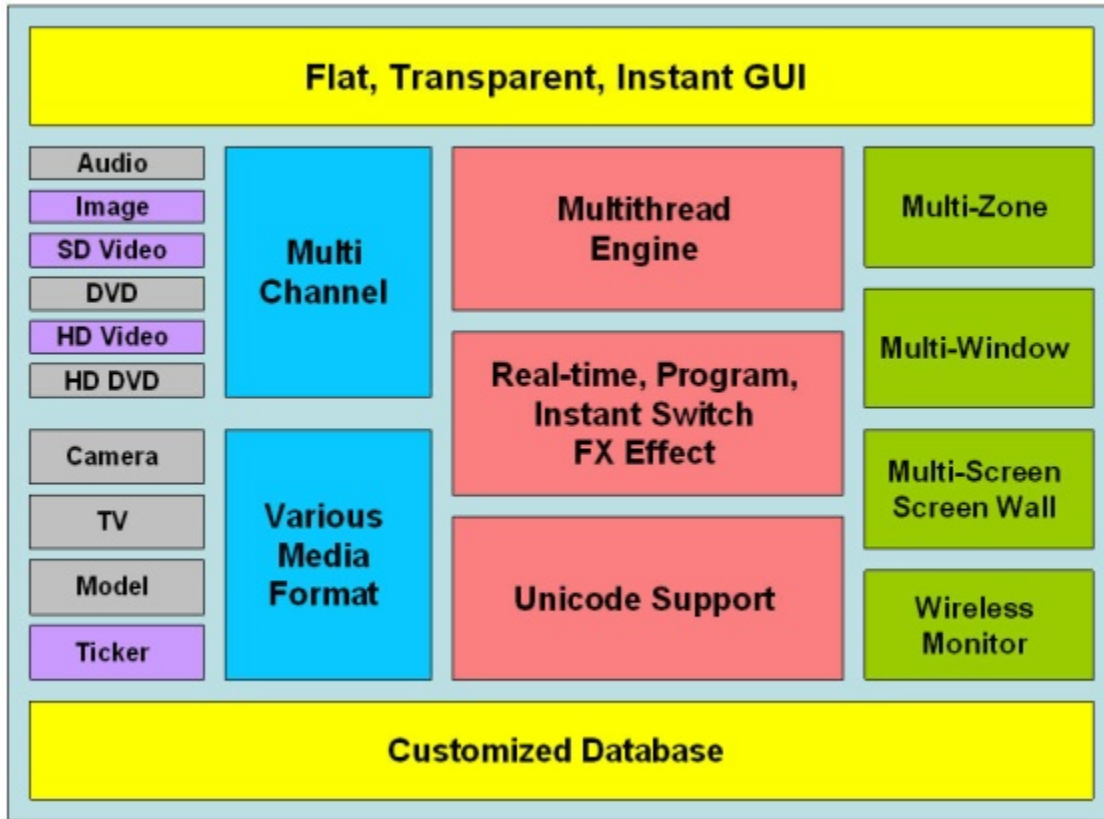
A TV program can be displayed in a channel.

14. Support DV camera, Web camera

DV camera, webcam can be used as a channel source.

Chapter 2 Architecture

The architecture of Adputer Composer consists of source, processing, and presentation. Below is the diagram of Adputer Composer architecture.



Diagram

In the source part, there are various media types and formats as well as multi-channels. Adputer Composer recognize standard video (SD) and high-definition video (HD) files. JPEG picture can also be a source file. In addition, a scroll text called ticker can be inputted instantly. The current available media types are tagged as purple color in the diagram. Other media sources such as audio, camera, tv, dvd player, and model are planed to be added in the future versions.

Adputer Composer recognizes video formats such as WMV, AVI, MPEG, and others. It can accept n channels simultaneously.

The pink blocks in the middle are processing units of Adputer Composer. There are Multi-thread Engine; Real-time, Programmable, Instant FX Effects; and Unicode Support.

These green blocks are belonging to presentation. It can support multiple zones, multiple windows, multiple screens, as well as wireless monitors.

Basically the **GUI of Adputer Composer** is flat. The use of controls is simple and obvious. The GUI controls are over the video surface, therefore users can control video playback over screen.

The top-left area of GUI shows the help and output message.

On the top-right area, there are toggled full screen button, open configuration database button, save configuration database button, help button, and language combo box.

At the bottom, the left area includes channel selection radio button, current channel playlist, and operations for playlist such as add/delete/up/down buttons.

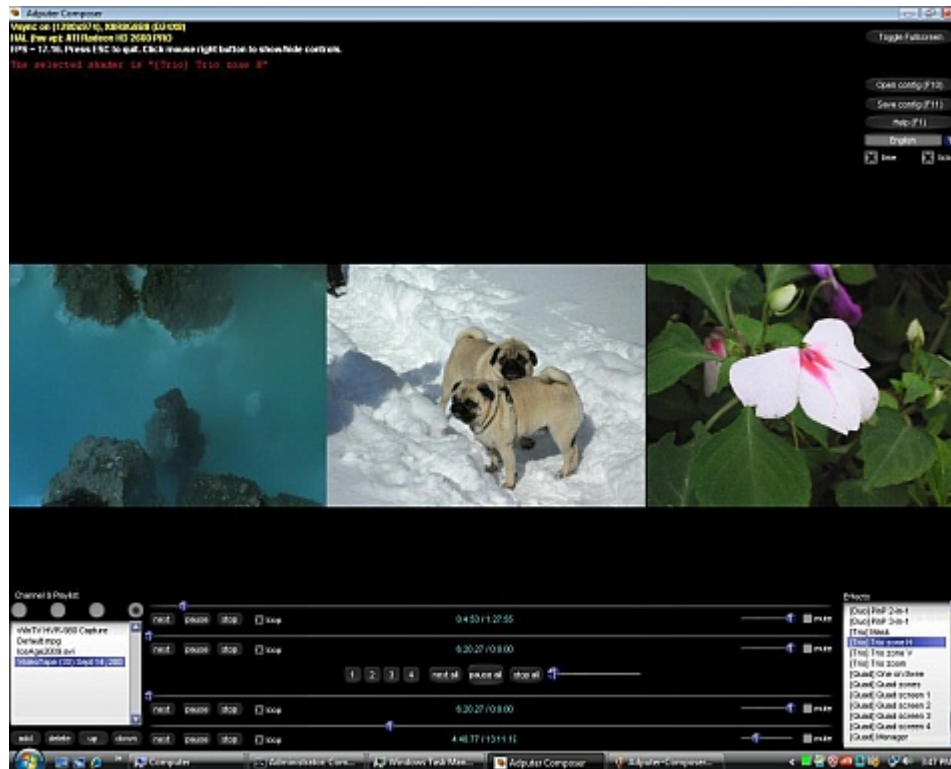
In the middle of the bottom, There are progress bar, video playback controls, volume and mute control for each channel. In the middle of the controls, there are substitution buttons, play/stop all buttons, and alpha bar. In the right of the bottom, it is the shader list.

At the bottom is the customized database that stores the configuration data and current states.

At the top is the Flat, Transparent, and Instant GUI module. We are going to discuss it in the following section.

Flat GUI

Below is a snapshot of **Aputer Composer** GUI. It is a flat, transparent, and instant GUI. Everything was designed to response instantly.



1) Vsync on (800x640) specifies that the refresh rate is synchronization with the monitor. The video resolution is 800x640. X8R8G8B8 is the format of video buffer. In the next line, HAL (hw vp) specifies that it is a hardware acceleration. NVIDIA GeForce Go 6150 is the name of graphics card. Furthermore, FPS (Frame Per Second) is an indicator of frame update speed.

A help message said that you can press **ESC** key to exit the player and you can click right mouse button to hide and show all controls. The next line of red text is the output message.

2) On the top-right area, there are toggled full screen button, open configuration database button, save configuration database button, help button, and language combo box.

Toggle Fullscreen allows you to switch the mode between full screen and normal window. You can click the **Open config (F10)** button to import a nwm database. You can click the **Save config (F11)** button to launch a Save File Dialog, then you can save current configuration of Aputer Composer into a **NeatWare Media**

file .nwm. Later you can launch Adputer Composer with .nwm file to resume your playback.

3) At the bottom, the left area includes channel selection radio button, current channel playlist, and operations for playlist such as add/delete/up/down buttons.

By double clicking an item in the playlist you can add a media file into the playlist of current channel. **File Open Dialog Box** is displayed after you double click an item in the playlist of current channel. You can also change the folder by clicking the drop-down list button on the top of the dialog box. To select a file you can click the file name in the list box and press the Open button.

Moreover you can import more media files in the dialog box into a playlist. You can press shift key down and move arrow keys to select a sequence of files or you can press ctrl key down and pick individual file by mouse clicking.

4) In the middle of the bottom, There are progress bar, video playback controls, volume and mute control for each channel. *Each channel has a Play/Pause button, a Stop button, a Loop check, a Volume scale, and a Mute check. You can move the indicator of the progress bar and stop the player on the selected channel.*

Below is more GUI controls. Each channel has an **Mute** check along with a timer indicator. The time is displayed in the format **mm:ss.aa** where mm is the minutes, ss is the seconds, and aa is the micro-seconds. The **current - duration** displayed the current time that a video is played and the duration is the length of a video. Note: The duration of a JPEG picture is 0.

In the middle of the controls, there are substitution buttons, play/stop all buttons, and alpha bar. For Adputer Composer there are n channel buttons for **channel substitution**. For example, you can click button 1 and then click button 3, the channel 1 and 3 will be exchanged. A math theory has proved that any substitution of a group of number can be completed by a series of exchange of a pair of numbers. By using substitution buttons we can define FX effects only on fixed and finite inputs.

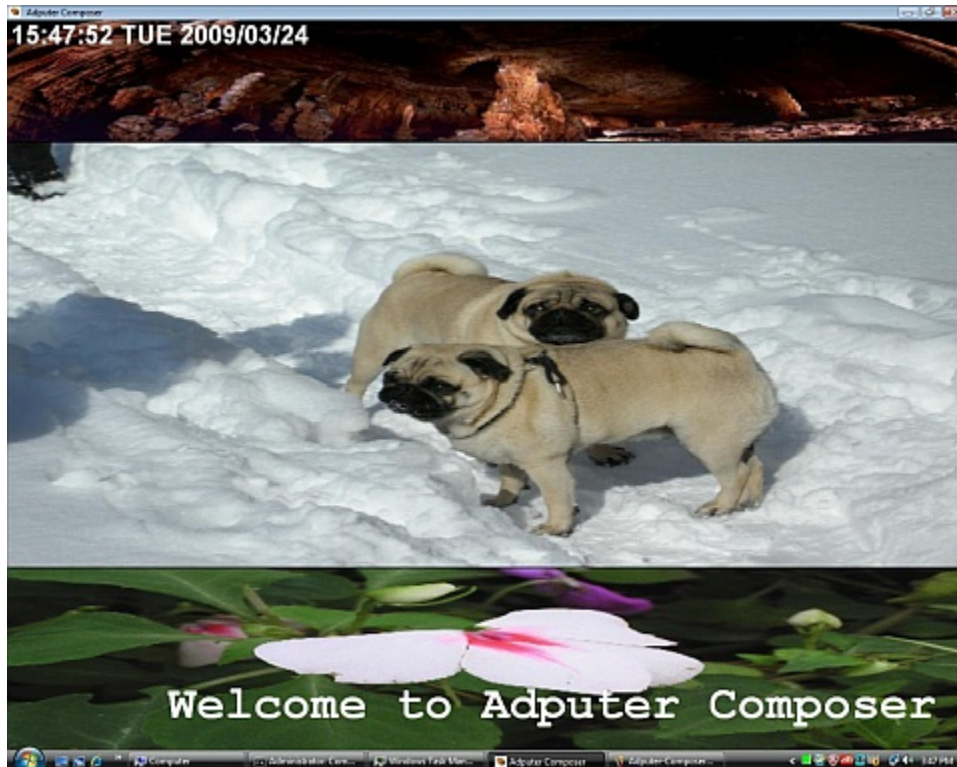
The **alpha** scale is useful to control effects as a parameters. For example, you can watch Fade effect by adjust alpha value. It is also been used to select a shader in a group.

You can click the **Play all/Pause all** button on the bottom right to start or pause the video playback for all channels. In the similar way, you can click **Stop all** button to stop all the video playback,rewind and replay from the original position.

5) In the right of the bottom, it is the shader list. Shaders are listed in a list box in the order of Solo, Duo, Trio, Quad Channel, and so on. You can select an item

by mouse clicking and scroll list up or down by mouse wheel.

Tips: you can pick the shader list and hide the control, then you can press the up and down arrow key to shuttle shader instantly. You will be surprised that the effects are showed in real-time.



6) Ticker allows you to use dynamic text in animation while playing video. You can insert or delete ticker text in the sqlite database manager. The check button allows you to hide or show a ticker when you hide the control.

Customization

You can customize Adputer Composer and store the configuration in an **.nwm** file. Later you can double click the **.nwm** or launch Adputer Composer in a command line in the following format:

"FullPath\Adputer Composer.exe" [FullPath\your.nwm] [-fullscreen]

If there is no **your.nwm** specification **Adputer Composer** will be launched along with the default nwm file stored in the data subdirectory of the installation. The parameter **full screen** specifies that the player will be launched in full screen.

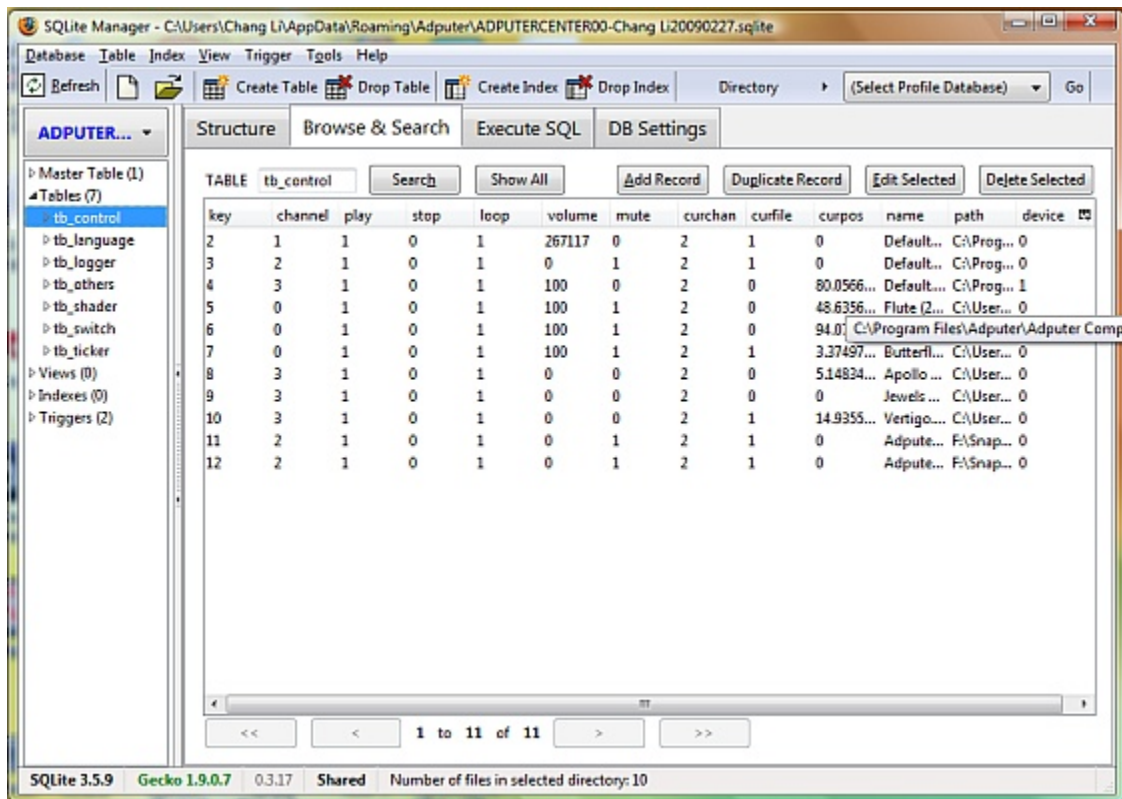
SQL Configuration Database Manager

Sqlite Manager is an add-on of Firefox browser that was used to manage configuration database in Adputer Composer. Below is the NWM file that consists of several SQL tables.

1) tb_control

tb_control is a table for media playback control of all the channels.

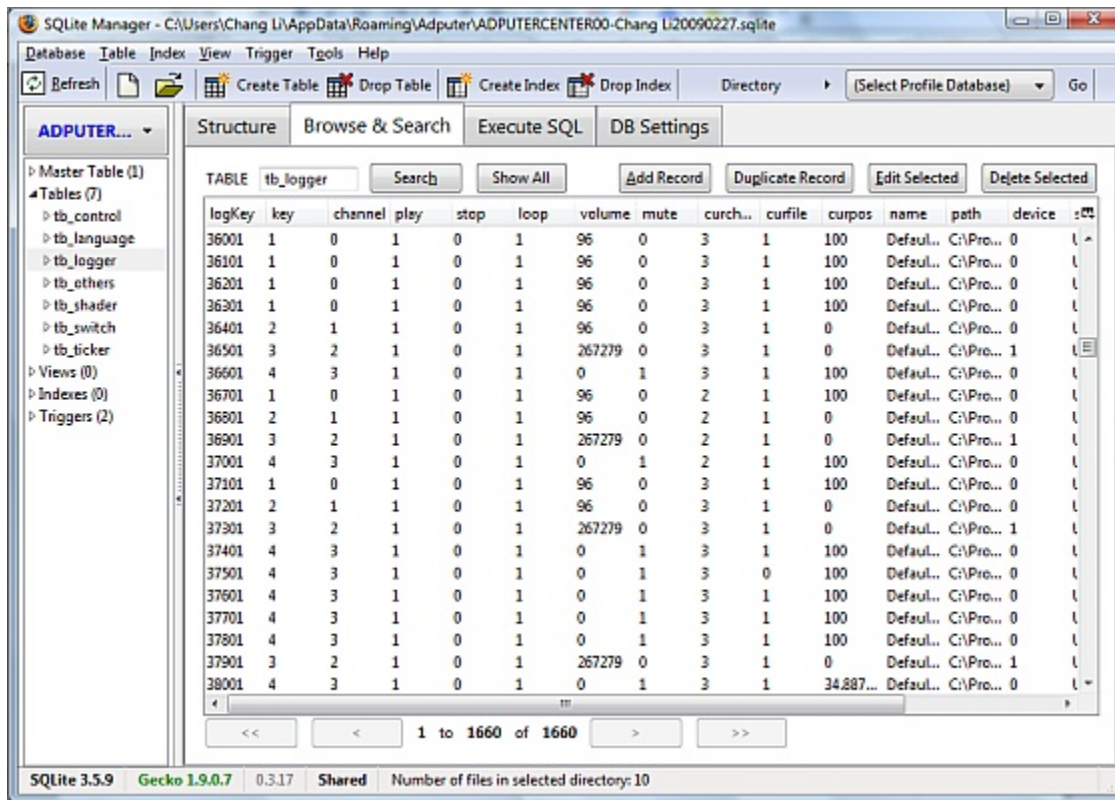
```
CREATE TABLE tb_control(
    key      INTEGER PRIMARY KEY,    // key
    channel  INTEGER,                // channel id from 0 to 3
    play     INTEGER,                // 1 for play, 0 for pause
    stop     INTEGER,                // stop state
    loop     INTEGER,                // 1 loop, 0 non-loop
    volume   INTEGER,                // volume from 0 to 100
    mute     INTEGER,                // 1 mute, 0 speak
    curfile  INTEGER,                // 1 is the current file, 0 not
    curpos   INTEGER,                // current video position
    name     TEXT,                  // media name
    path     TEXT                    // media path
)
```



2) tb_logger

tb_logger records the date and time of the upgrade in database.

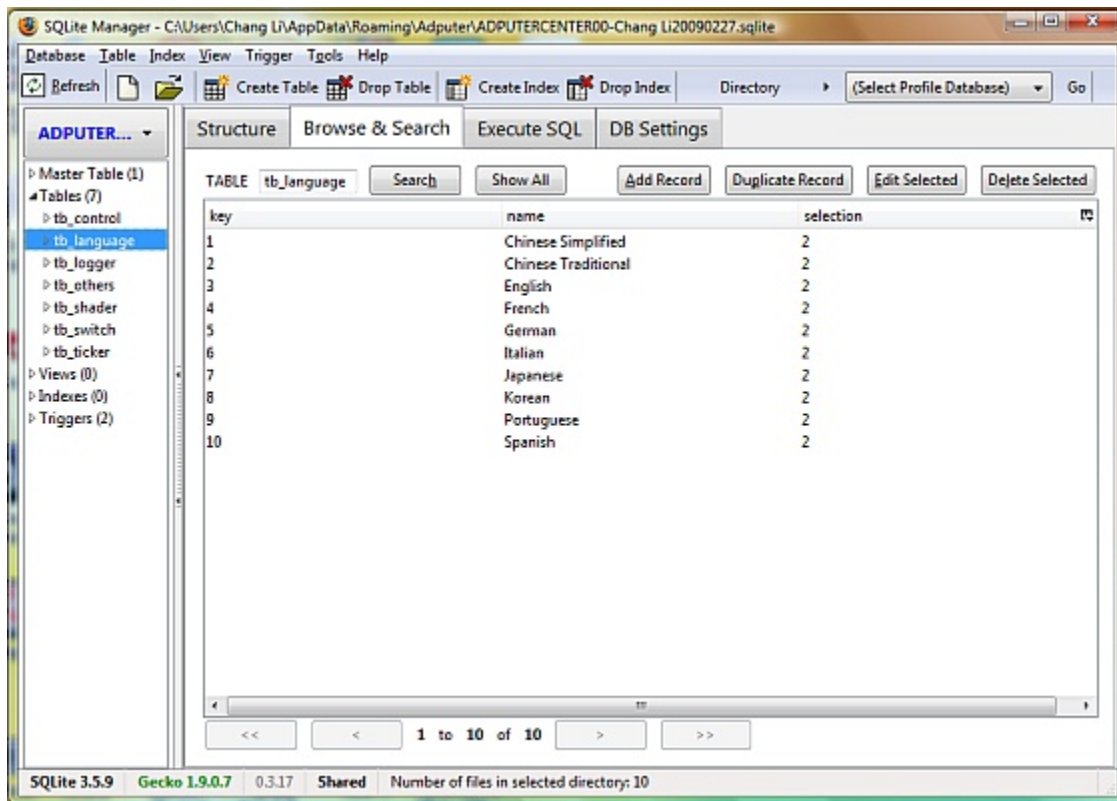
```
CREATE TABLE tb_logger(
    logKey    INTEGER,           // logger key
    key       INTEGER,
    channel   INTEGER,
    play      INTEGER,
    stop      INTEGER,
    loop      INTEGER,
    volume    INTEGER,
    mute      INTEGER,
    curfile   INTEGER,
    curpos    REAL,
    name      TEXT,
    path      TEXT,
    sqlAction VARCHAR(32),      // sql action
    DateEnter DATE,           // date
    TimeEnter TIME            // time
)
```



3) tb_language

tb_language is the table for language selection.

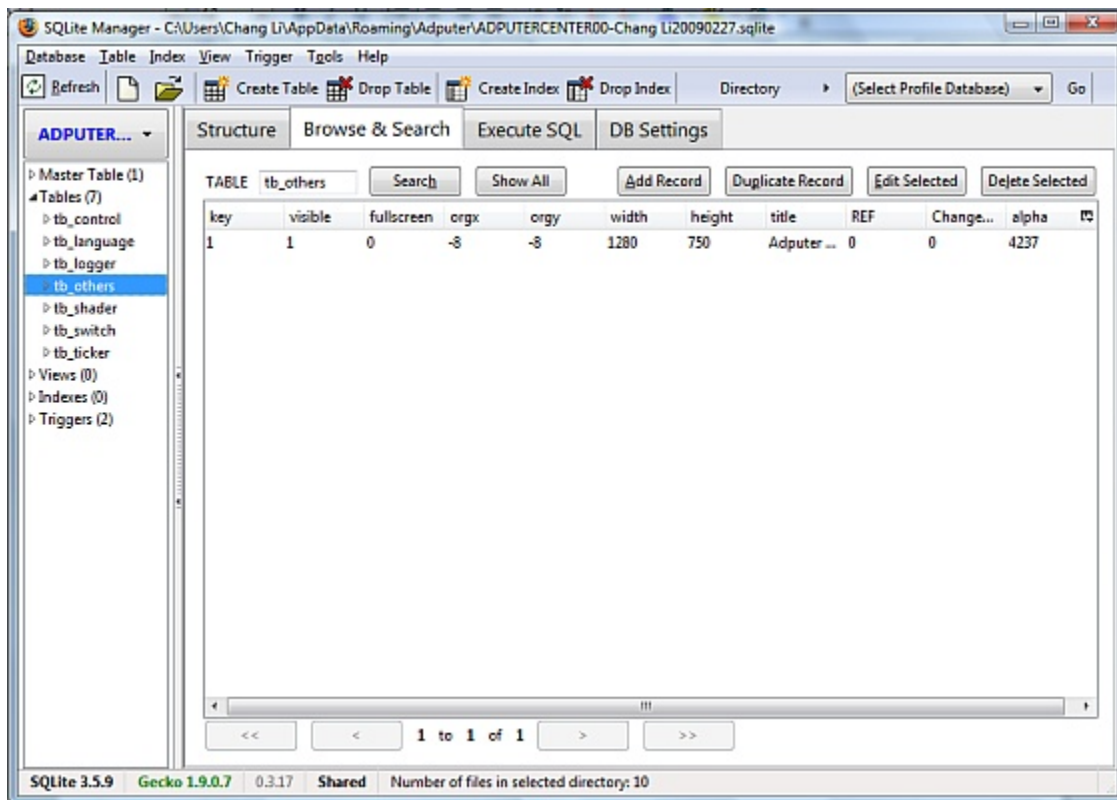
```
CREATE TABLE tb_language(
    key          INTEGER PRIMARY KEY,      // key
    name         TEXT,                    // language name
    selection    INTEGER                    // selected item
)
```



4) tb_others

tb_others is the table for window's control.

```
CREATE TABLE tb_others(
    key          INTEGER PRIMARY KEY, // key
    visible      INTEGER,           // 1 visible, 0 invisible
    fullscreen   INTEGER,           // 1 fullscreen, 0 windows
    orgx         INTEGER,           // orgx
    orgy         INTEGER,           // orgy
    width        INTEGER,           // width
    height       INTEGER,           // height
    title        TEXT,              // window's title
    REF          INTEGER,           // REF for debug
    ChangeDevice INTEGER,           // for debug
    alpha        INTEGER            // parameter 0.0 to 1.0
)
```



5) tb_shader

tb_shader is the table for shader control. For Adputer Composer Developers, you can insert, upgrade, or remove an item of shader in the list. When you designed a new HLSL shader in the .fx file, you need to add a new item with the corresponding *shader_name* in the .fx file. The *list_name* is for the display in the shader list box.

```
CREATE TABLE tb_shader(
    key          INTEGER PRIMARY KEY, // key
    list_name    TEXT,                // shader list name
    shader_name  TEXT,                // shader name
    selection    INTEGER              // selected item id
)
```

The screenshot shows the SQLite Manager interface with the 'tb_shader' table selected. The table structure is as follows:

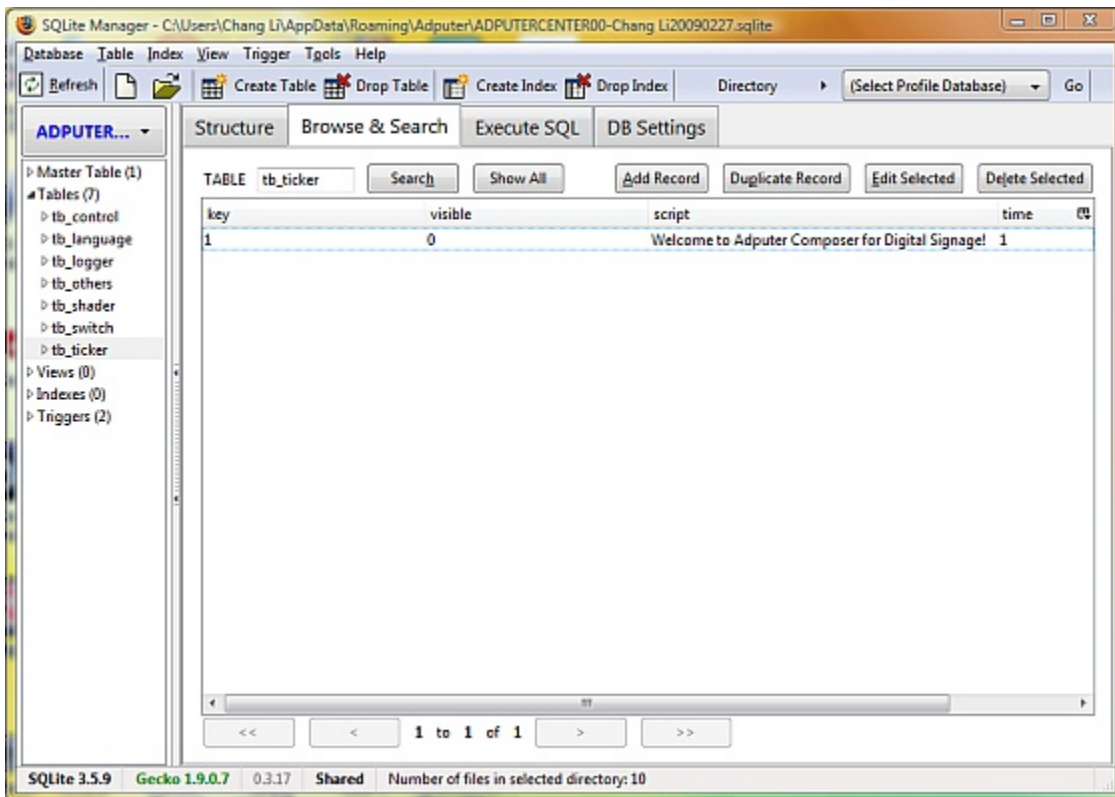
key	list_name	shader_name	selection
1	[Solo] CH1 show	SoloChannel_1	0
2	[Solo] CH2 show	SoloChannel_2	0
3	[Solo] CH3 show	SoloChannel_3	0
4	[Solo] CH4 show	SoloChannel_4	0
5	[Solo] Turn left 90	SoloChannelYX_90	0
6	[Solo] Turn left 180	SoloChannelYX_180	0
7	[Solo] Turn right 90	SoloChannelYX_270	0
8	[Solo] Turn by alpha	SoloChannelYX_Alpha	0
9	[Solo] Negative	SoloChannel_Negative_1	0
10	[Solo] Black white	SoloChannel_BlackWhite_1	0
11	[Solo] Bin threshold	SoloChannel_BinaryThreshold_1	0
12	[Solo] Sepia tone	SoloChannel_SepiaTone_1	0
13	[Solo] Edge	SoloChannel_Edge_1	0
14	[Solo] Noise	SoloChannel_Noise_1	0
15	[Solo] Blur	SoloChannel_Gaussian_1	0
16	[Solo] RGB To GBR	SoloChannel_RGB2GBR_1	0
17	[Solo] RGB To GRB	SoloChannel_RGB2GRB_1	0
18	[Solo] RGB To RBG	SoloChannel_RGB2RBG_1	0
19	[Solo] RGB To BGR	SoloChannel_RGB2BGR_1	0
20	[Solo] Cont. threshold	SoloChannel_ContinuousThres...	0
21	[Solo] Bilinear	SoloChannel_Bilinear_1	0

The interface also shows a sidebar with a tree view of the database structure, including tables like tb_control, tb_language, tb_logger, tb_others, tb_shader, tb_switch, and tb_ticker. The status bar at the bottom indicates 'SQLite 3.5.9', 'Gecko 1.9.0.7', '0.3.17', 'Shared', and 'Number of files in selected directory: 10'.

6) tb_ticker

tb_ticker is the table for ticker processing.

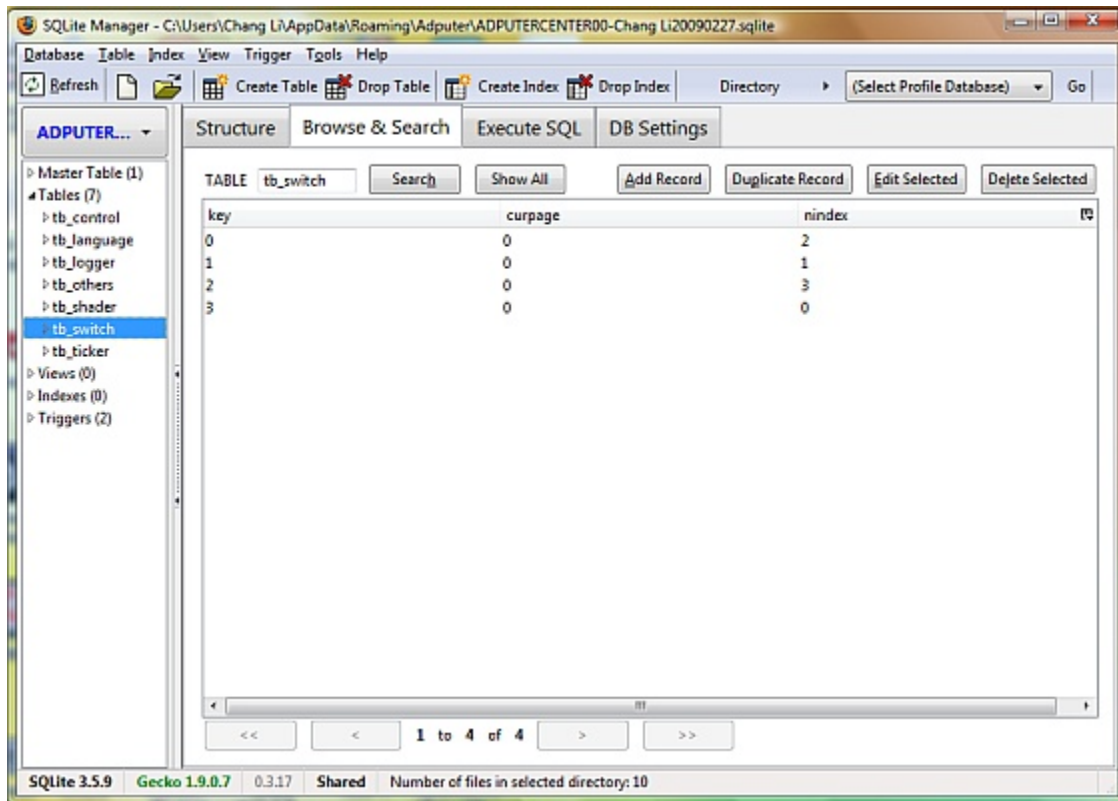
```
CREATE TABLE tb_ticker(
    key      INTEGER PRIMARY KEY,    // key
    visible  INTEGER,                // 1 visible, 0 invisible
    script   TEXT                    // ticker script message
)
```



7) tb_switch

tb_switch is the table for channel swap.

```
CREATE TABLE tb_switch(  
    key      INTEGER PRIMARY KEY,    // key  
    curpage  INTEGER,                // current page 0  
    nindex   INTEGER                 // channel index from 0 to 3  
)
```



Keyboard Control

Aputer Composer also provides keyboard control for video playback. In default, F1 is for help, F2 for play/pause all, F3 for stop all, F4 for ticker show/hide, F11 for .nwm file saving. You can press ESC key to exit program. You can also switch to and back from full screen by pressing ALT + Enter key.

Chapter 3 Technologies

High-Definition Video

The resolution of a video can be defined as Low Definition (**LD 240p**), Standard Definition (**SD 480p**), Normal High-Definition (**HD 720p**), Full High-Definition (**HD 1080p**), and Super High-Definition (**SHD 2060p**). Usually, the number of visible (scan) lines, that is the height of a digital picture, is the index of definition.

The resolution of *Low Definition (LD)* is from 128p to 240p where **p** represents the progressive scanning. In contrast, the low case letter **i** represents the interlace scanning. LD display usually applied in mobile phones and handheld devices.

The resolution of *Standard Definition (SD)* is from 320p to 480p. Most of TVs and DVD are belong to the SD resolution. Normally, the ratio aspect of SD Video is 4:3.

The resolution of *Normal High Definition (HD)* is defined as 1080/i and 720p. The resolution of *Full High-Definition* is defined as 1080p. The ratio aspect of HD Video is 16:9.

The resolution of *Super High-Definition (SHD)* is beyond 1080p, which can be 1440p, 1800p, 2160p, 2520p, and 2880p. So far there is no standard for SHD.

Today HD technologies have made HD media affordable and available.

- HD Camera**

HD digital video camera can shot in 16:9 aspects rate directly.

- HD Display**

Large and wide screen monitor from LCD, Plasma, and Digital TV technology.

- HD Processor**

Low power and fast processor can be the codec to encode and decode digital video such as MP4.

- HD Sound**

Stereo sound evolutes from 2.1 to 7.1. The MP3 has been widely used to

replace the CD quality music. New Dolby and DTS have been selected to be the HD sound.

HD Storage

LD media can be stored in CDROM, CDR, or CDRW. Its capacity is 680MB. SD media such as DVD are stored in the DVD-ROM, DVD-R, or DVD-RW. Its capacity is 2GB or 4GB. New generation HD media will be stored in the Blue Ray or HD DVD. Its capacity is 20GB.

HD Network

HD video can be transmitted through broadband or 3/4G mobile wireless network in downloading or streaming method.

HD TV

High-Definition TV can be transmitted by cable or digital satellites.

HD Player

Many devices such as DVD, PC, Console, Set-top box can play HD videos.

There are many formats for High-Definition video. Microsoft's WMV HD is one of them. However, most of standards are based on MPEG standard.

MPEG means Moving Picture Experts Group. This group is working under the direction of International Standards Organization (ISO) for the standards of compressed motion pictures and associated audio.

MPEG-1 had been completed on October 1992 known as International Standard ISO-11172. Video CD (VCD) adapted MPEG-1 as its video standard for CD-ROM media. VCD's audio adapted MP2 audio. The normal MPEG-1 video resolution is **352x240** or **240p**.

DVD adapted the MPEG-2 as video standard and AC3 as audio standard. The normal MPEG-2 video resolution is **704x480** or **480p**. Usually the movie data of DVD is stored in the .vob file.

MPEG-4 or MP4 is the latest MPEG standard for low-bit-rate streaming video. It is a standard that is suitable for wireless multimedia and broadband applications. Its resolution can be **480p**. Compare to DVD video, MP4 requires smaller space with the same picture quality.

MP4 HD is a standard to apply MPEG-4 video on High Definition video. Its resolution is defined as **720p** or **1080p**.

Multiple Channel Video Mixing

Today most of real-time video mixing is done on special hardware machines. Those machines are expensive and non-flexible. Unfortunately current general purpose Central Processing Unit (**CPU**) is not fast enough to complete multiple video composition in real-time.

The rapid development of the Graphics Processing Unit (**GPU**) technology had made the fastest GPU to be 10 or 50 times faster than the fastest CPU. The power of GPU with its parallel streaming pipelines makes real-time video mixing on GPU possible.

In theory CPU is good at procedure control and GPU is good at stream computing. **Adputer Composer**, powered by its multi-thread media engine and GPU shader can implement multiple channel video composition in real-time.

The media engine of **Adputer Composer** is benefit from modern technologies of CPU, GPU, and Chipset. The dual-core CPU from Intel and AMD will speed up the processing of multiple video streams in codec. Multi-core processor can speed up the multi-thread codec in Adputer Composer naturally.

Furthermore, the real-time video effects can be completed on GPU by the shader library of Adputer Composer.

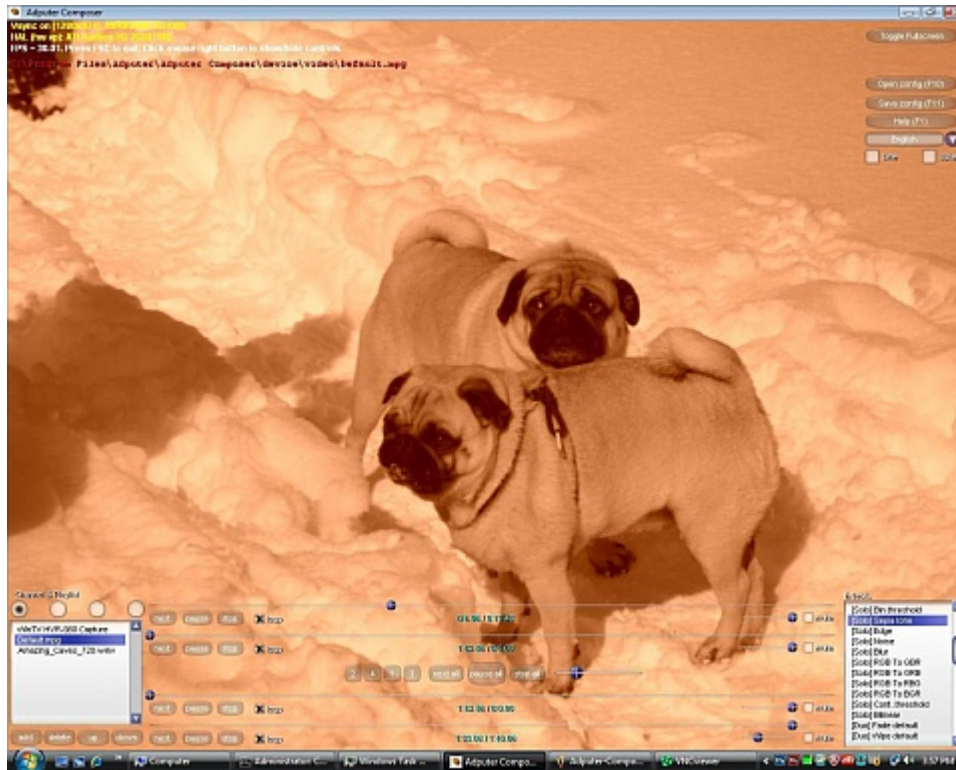
Finally, multiple GPU processors as SLI/Crossfire can greatly improve the performance of video mixing on HD Media. The fast data channels of DRAM, PCIe Bus, and SATA hard disk are also speed up video mixing.

Video mixing, sometimes called *cinematic composition*, is to mix multiple video streams, do processing, and generate an output. Video Mixing can be represented as a multiple variable function, $R = S(C_0, C_1, \dots, C_{n-1})$ where C_i is the Channel i of video stream, S is the Shader, and R is the Render. In GPU processing, a video stream is set as a texture.

Now let's look at the video mixing by the number of channels.

1) Solo Channel

Similar to image processing, only one single video stream as input can do transform based on each video frame. There are many effects such as black and white, negative, and edge.



Negative is the reverse of the input video colors. Black and White maps color to gray levels. There are Sepia Tone, Edge, Bilinear Blur, Gaussian Blur, Median, and Color Swap

2) Duo Channel

Video mixing from two or more channels is often called *video composition*. Usually, a video channel is treated as foreground video and another video channel is considered as the background video. We are going to call them foreground channel and background channel respectively. A special case is that the background video can be a picture.

- **Chromic Key Effect**

The famous **chromic key** processing in the traditional video mixing can be easily implemented in a GPU shader.

Suppose the foreground video such as an actor is shot on a background of green screen, and the background video is the scene of a beach. So the value of chromic key is the green color. Duo channels can be composited in the following way: open foreground channel when its pixel value is not equal to the value of chromic key and open background channel otherwise. This concept can be represented as a C conditional expression.

```
result = (color0 == key) ? color1 : color0;
```

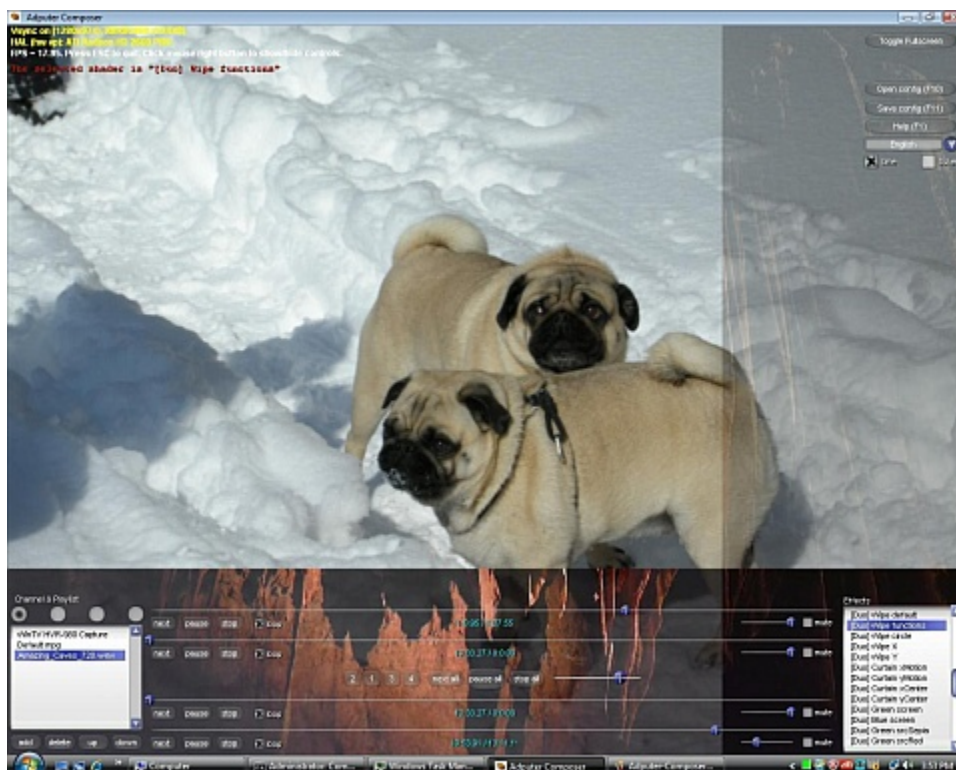
-

- **Fade Effect**

Another composition is called **fade**. In this case a control variable alpha in [0.0, 1.0] is applied to generate a linear combination of two channels. Fade is the sum of *alpha* multiples the pixel value of first channel and *1.0-alpha* multiples the pixel value of second channel.

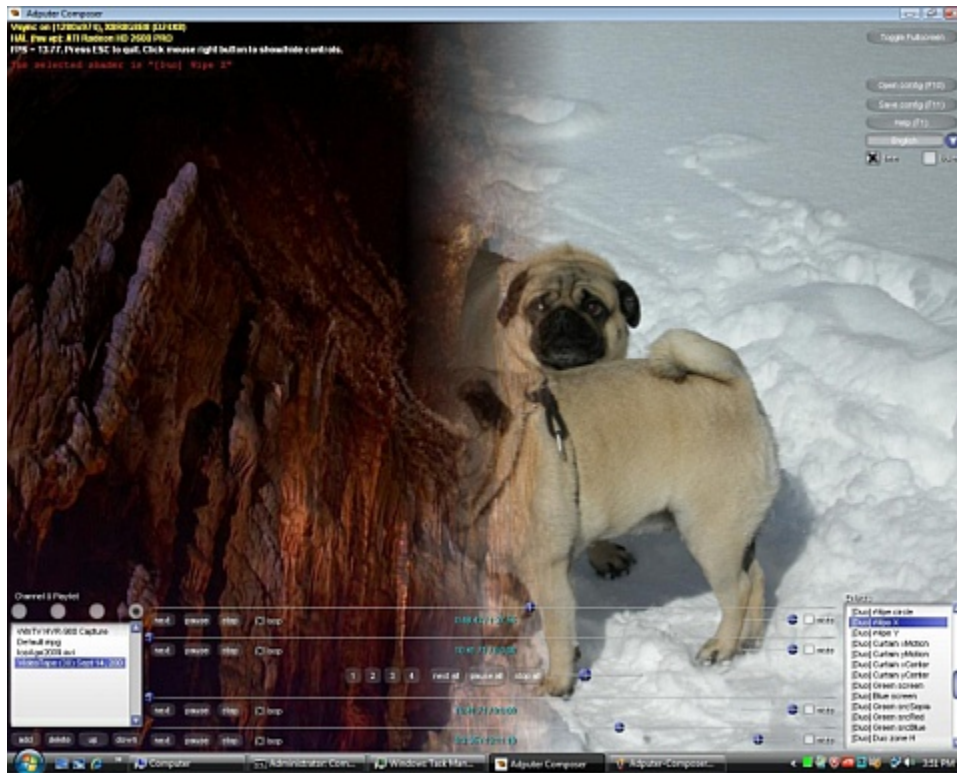
The HLSL code of fade is represented below. color0 and color1 are the pixel values of channels respectively. And the alpha is the transparency value.

$$\text{result} = \text{alpha} * \text{color0} + (1.0 - \text{alpha}) * \text{color1};$$



- **Wipe Effect**

In addition, the *wipe* composition is widely used in movie effects. It is based on the position of pixel of a video frame. Wipe transform sweeps frame B over the frame A from left to right.



There are three rectangle regions: frame B only on the left, gradient region on the middle, and frame A only on the right. The gradient region is the rectangle mixing between frame A and B. The boundary between frame A region and the gradient region is called the **leading edge**. Similarly, the boundary between the frame B region and the gradient region is called the **trailing edge**.

Finally, the value of *alpha* can be the function of time. For example, a sin (time) alpha value can display two channels alternatively and smoothly from one channel fade in and fade out to another channel. The HLSL sample code is

```
alpha = saturate( sin( fTime ) + 1 ) / 2.0;  
result = alpha * color0 + ( 1.0 - alpha ) * color1;
```

- **Frame Effect**

A headline video is mixed with an illustration video by wipe effect in special position. The headline video can be in the center of the display while the illustration video can be displayed around the headline video like a frame. A signature can be overlay on the bottom of the headline video as a watermark. The videos can be replaced as pictures.



In this example, the headline picture is two dogs. The illustration is a high resolution video. The FX effect is circle wipe transform.

- **Curtain Effect**

Curtain effect is the mixing of a picture and a video or another picture while the first picture moves like a curtain to cover the second media gradually. Curtain effect makes a sense of mysterious to audiences who may be attracted by the disclosure of content gradually. And audiences may pay longer time to watch the video.

The procedure to generate a curtain ads is simple in Adputer Composer:

- i) **Generate curtain media**

Curtain media can be a picture or a video. Usually a picture is used to show a logo or a promotion text, while a video is a decoration to ads.

- ii) **Generate main media**

Execute Adputer Composer

- iii) **Select curtain and main media**

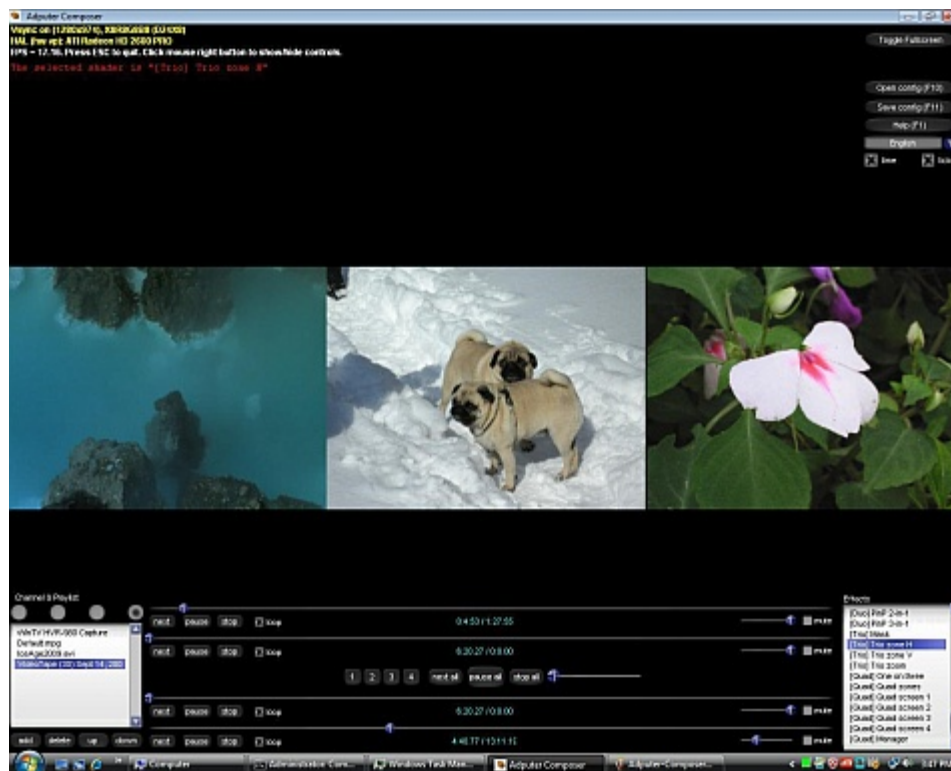
The curtain video is disclosed from left to right. In the end the curtain video disclosed from right to left.

3) Trio Channel

A well known example of trio channel is the **mask** transform. A mask is a black and white image. Two video channels and a mask picture channel are the inputs. The effect is the sum of the first channel multiplies the mask and the second channel multiplies the negative of mask. Therefore, two channels are outputted in the areas distinguished by the shape of the mask.

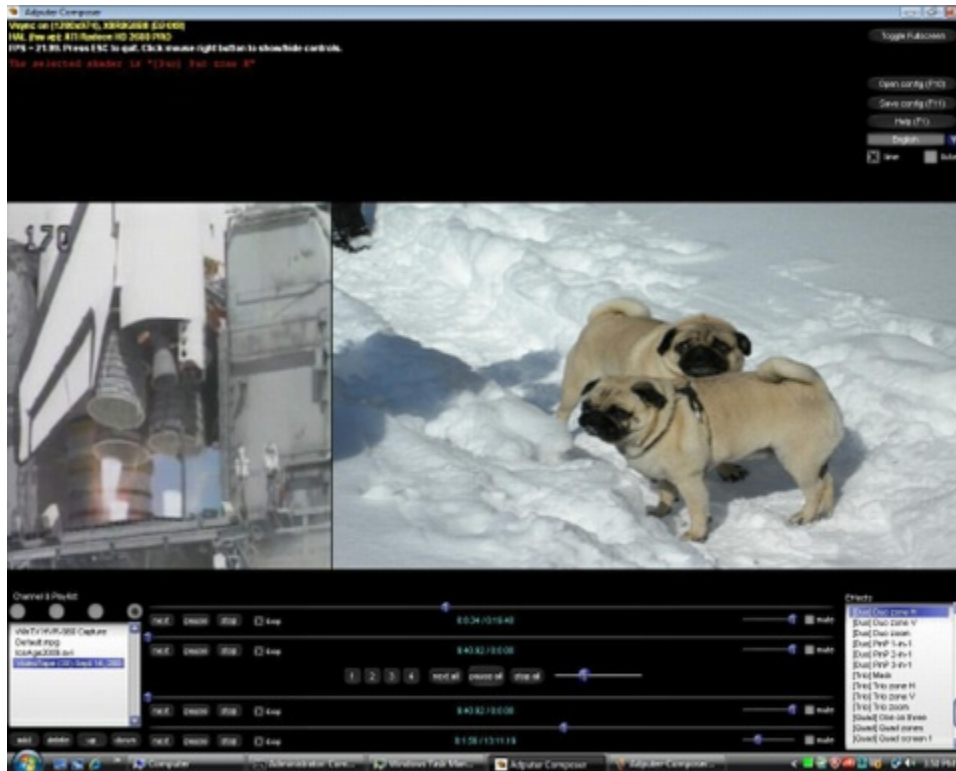
The HLSL sample code is simple:

result = mask * color0 + (1.0- mask) * color1;



4) Multiple Channels

More video mixing can be considered as multiple layer composition. The lighting and motion effects can make a common video uncommon. For example, a channel that shows a light source from left to right and another video that shows another light source from right to left can be composited along with a channel to generate dynamic lighting effects. This technology has been widely used in advertising and movie trailers.



HLSL Shader

HLSL language can help you to write a shader easily. Some media players lack of shader support or use a plug-in running on CPU for video mixing. Compare to HLSL just-in time compiler inside the Adputer Composer, a plug-in solution for video mixing is hard to build because you may be required to write C programs for CPU.

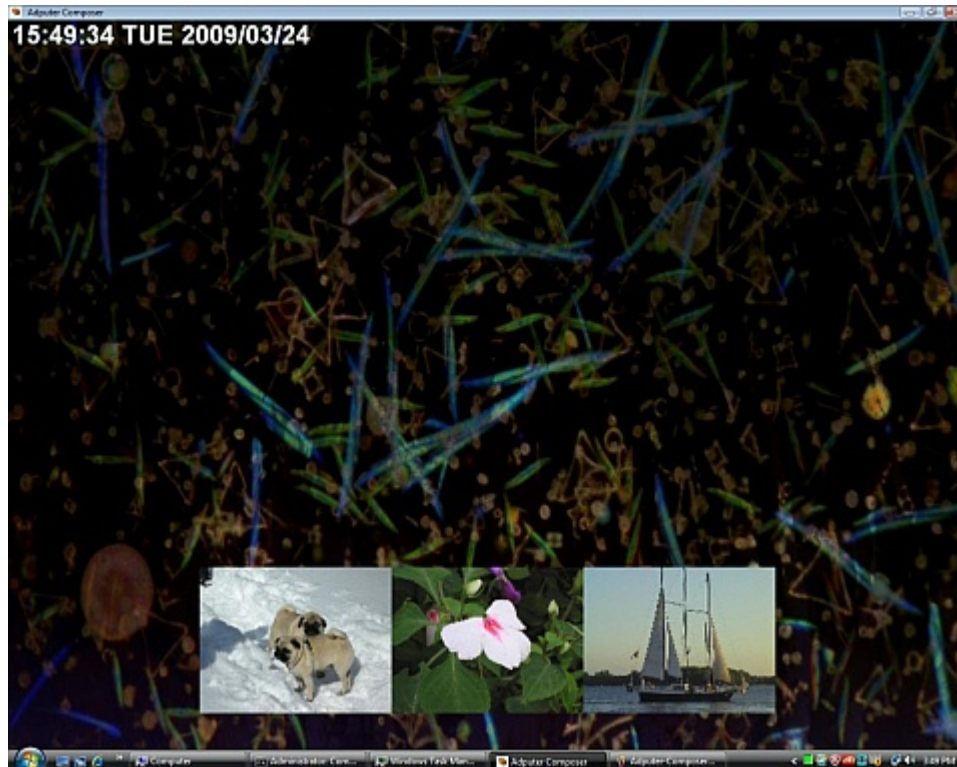
High Level Shading Language (HLSL), a programming language for Graphic Processing Unit (GPU) in DirectX 9.0 and plus, supports the shader construction with C-like syntax, types, expressions, statements, and functions.

Long time ago, Apple's *RenderMan* was a popular shading language that was used to generate cinematic effects with CPU in render farms. Lately, Microsoft's High-Level Shading Languages (*HLSL*) and OpenGL's OpenGL Shading Language (*GLSL*) shading languages have been developed for real-time shader on GPU. Best integrated into the DirectX 9, HLSL works solely on Windows platform. Similarly, OpenGL 1.5 starts to include OpenGL shading language GLSL as a standard component. These high level languages accelerated the shader development.

To build a complete shader, new shading languages for GPU must work along with a host programming language such as C++. Other languages can also be used to build shader. However, although the procedure is tedious to set large amount of parameters, C++ is fastest. Adputer Composer is a video player that supports DirectX 9.0+, HLSL, and Shader Model 3.0. Developers can design their own HLSL pixel shader for real-time video post-processing with Adputer Composer.

The Simplest Example

A shader consists of vertex shader and pixel shader. The stream of 3D model flows from application to the vertex shader, then to the pixel shader, finally to the frame buffer. `a2v` struct represents the data structure transferred from an application to a vertex shader, `v2p` from a vertex shader to a pixel shader, and `p2f` from a pixel shader to the frame buffer. Below program transforms a vertex's position into the position of clip space by view matrix.



```
struct a2v {
    float4 Position : POSITION;
};
```

Inside the HLSL function, struct `a2v` specifies a vertex structure that represents the data of vertices.

```
struct v2p {
    float4 Position : POSITION;
};
```

struct `a2v` specifies a stream structure from vertex to pixel shader. Position is a four dimensional vector declared by float4. Furthermore, POSITION, called output semantic, indicates the initialized type of Position.

```
void main( in a2v IN, out v2p OUT, uniform float4x4 ModelViewMatrix )
{
    OUT.Position = mul( IN.Position, ModelViewMatrix );
}
```

main is a vertex shader function name. void means that this function will return nothing. The input parameter IN is specified by in modifier and struct a2v. Similarly, the output parameter OUT is specified by out modifier with the type v2p. In addition, float4x4 declares a matrix ModelViewMatrix. The uniform modifier indicates that the value of the matrix is a constant assigned by external program.

Finally, this simplest vertex shader outputs the multiplication of the vector Position and the matrix ModelViewMatrix. While IN.Position is the left parameter of mul, it is considered as a row vector. Otherwise it is considered as a column vector.

HLSL provides scalar data type like float and vector data type like float3. The scalar data types include bool with true or false value, int with 32-bit signed integer value, half with 16-bit floating point value, float with 32-bit floating point value, and double with 64-bit floating point value. A shader emulation will work while a GPU does not support a data type.

A vector data type is declared as `vector<type, size>` where size is the dimension and type is the scalar component type. vector is a declaration of 4 dimensional float vector. Usually, we use `float2`, `float3`, and `float4` for two, three, and four dimensional float vectors.

float4x4 declares a float matrix type with 4 rows and 4 cols. A general matrix declaration has the form `matrix<type, row_size, column_size>`. A variety of dimensional declaration such as `float3x4` is also acceptable. To access an element of matrix you can use `m[i][j]` or zero-based row-column position like `_m00` as well as one-based row-column position like `_11`.

You can think HLSL as a C language for GPU programming except there are no pointer, union, bitwise operations, and function variables. There are no goto, switch, recursive function in HLSL as well. However HLSL adds vector data type, build-in constructor, `swizzling` and masking operators. HLSL standard library includes mathematical functions and texture processing functions. The function overloading has been used to unify the operations of different vectors.

```
vs_1_1
dcl_position v0
m4x4 oPos, v0, c0
mov oD0, c4
```

This **asm** program is the compiled code of the simplest HLSL example. First, `vs_1_1` specifies the version of vertex shader as 1.1. Second, `dcl_position v0` declares that `v0` is a position register. The third statement declares a matrix multiply with source variable register `v0` and constant register `c0` where `oPos` represents the destination position register. Finally, the last statement moves the value of register `c4` to register `oD0`. Usually, `vN` represents input register and `oXXX` represents output register in the assembly vertex shader language.

Pixel Shader

Pixel shader completes the computing of pixels.

```
float brightness;
sampler2D tex0;
sampler2D tex1;
```

brightness is the value to control the bright of light. Both tex0 and tex1 are the sampler of the textures.

```
struct v2p {
    float4 Position : POSITION;
    float2 Texcoord0 : TEXCOORD0;
    float2 Texcoord1 : TEXCOORD1;
    float4 Color    : COLOR0;
};
```

v2p declares a struct type that transfers data from vertex shader to pixel shader. It is the same as the struct in vertex shader. The input semantics of pixel shader can be $COLOR_n$ for Color or $TEXCOORD_n$ for Texture coordinates. Although the struct v2p must be the same as the v2p in the vertex shader, the item Position can not be read in the pixel shader, because it is not bind by the input semantics of the pixel shader.

```
struct p2f {
    float4 Color : COLOR0;
};
```

p2f declares output data structure and OUT is the output object. The output semantics of pixel shader can be $COLOR_n$ of Color for render target n and/or DEPTH for Depth value.

```
void main( in v2p IN, out p2f OUT )
{
```

Constant parameter brightness has a float type. [sampler2D](#) specifies a 2D texture unit. When you plan to access a texture you must use sampler with an intrinsic function. A sampler can be used for multiple times.

```
float4 color      = tex2D( tex0, IN.Texcoord0 );
float4 bump       = tex2D( tex1, IN.Texcoord1 );
```

fetch texture color and bump coordinate for further computing of bump effect. [tex2D](#) is an texture sampling intrinsic function of HLSL. It generates a vector from

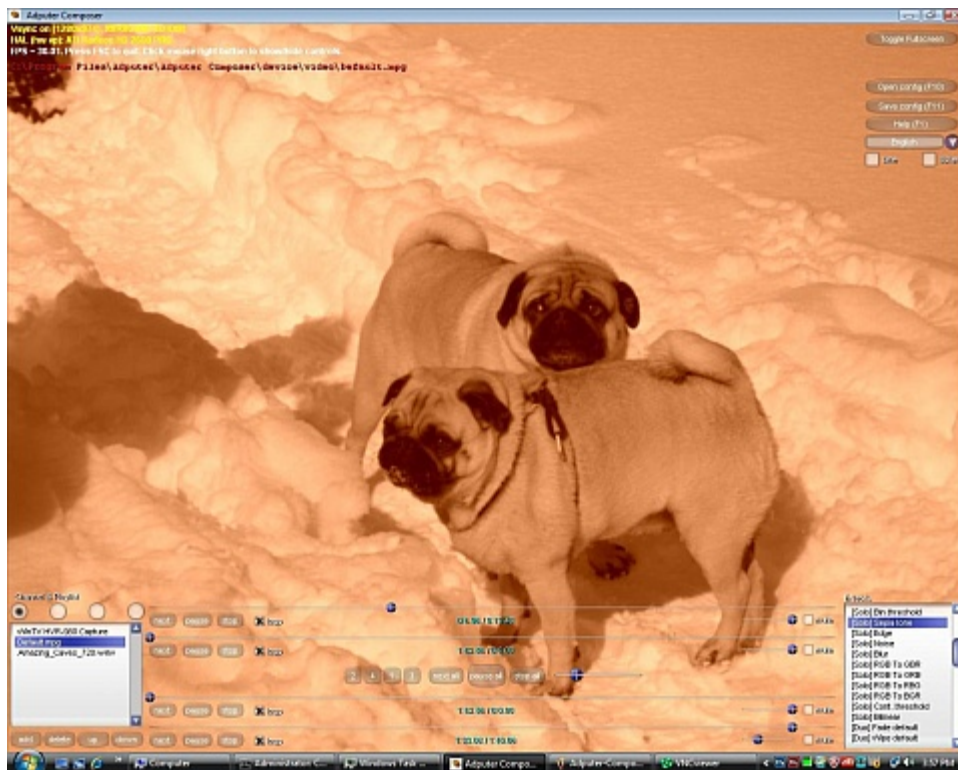
a texture sampler and a texture coordinate.

```
OUT.Color = brightness * IN.Color * color;
}
```

the code multiplies *brightness*, *IN.Color* and *color* to generate output RGBA color vector.

Image Processing: Sobel Edge Filter

This program shows the implementation of Sobel edge filter with a HLSL pixel shader. In the similar way we can implement many image filters in pixel shader.



```
float Brightness;
sampler2D tex0;
```

global variables

```
struct v2p {
    float4 Position : POSITION;
    float2 Texcoord : TEXCOORD0;
    float4 Color : COLOR0;
};
```

```

struct p2f {
    float4 Color    : COLOR0;
};

```

main function includes a vertice struct as input, a float parameter as brightness control, and a 2D texture sampler. The return value has float4 type with the semantic COLOR0.

```

void main( in v2p IN, out p2f OUT )
{

```

const specifies the constants. The c[NUM] is a float2 constant array. Notes its initialization is convenience like C language. col[NUM] is a variable array of type float3 with NUM elements. int i declares the i as integer. This usage is effective for pixel shader 2.0 or later.

```

    const int NUM = 9;
    const float threshold = 0.05;

    const float2 c[ NUM ] = {
        float2(-0.0078125, 0.0078125),
        float2(0.00, 0.0078125),
        float2( 0.0078125, 0.0078125),
        float2(-0.0078125, 0.00 ),
        float2( 0.0,      0.0),
        float2( 0.0078125, 0.007 ),
        float2(-0.0078125,-0.0078125),
        float2( 0.00 ,   -0.0078125),
        float2( 0.0078125,-0.0078125),
    };

    float3 col[ NUM ];
    int i;

```

it stores the samples of texture to col array.

```

    for (i=0; i < NUM; i++) {
        col[i] = tex2D( tex0, IN.Texcoord.xy + c[i] );
    }

```

now we start to compute the luminance with dot product and store them in lum array.

```

    float3 rgb2lum = float3(0.30, 0.59, 0.11);

    float lum[ NUM ];

```

```
for (i = 0; i < NUM; i++) {  
    lum[i] = dot(col[i].xyz, rgb2lum);  
}
```

Sobel filter computes new value at the central position by sum the weighted neighbors.

```
float x = lum[2]+ lum[8]+2*lum[5]-lum[0]-2*lum[3]-lum[6];  
float y = lum[6]+2*lum[7]+ lum[8]-lum[0]-2*lum[1]-lum[2];
```

it shows the points which values are over the threshold and hide others. Final result is the product of col[5] and edge detector value. Brightness adjusts the brightness of the image.

```
float edge =(x*x + y*y < threshold)? 1.0:0.0;
```

final output

```
OUT.xyz = Brightness * col[5].xyz * edge.xxx;  
OUT.w   = 1.0;  
}
```

FX Effects

Effects Framework

To create a set of visual effects in a logical unit, Microsoft defined effects framework. Effect files are referred to as **fx** files with the extension **.fx**. It can be compiled as a **.fxo** file. Adputer Composer requires SM2.0 support. The structure of an **fx** file looks like the following:

Global Variables

Effect States

Pipeline, Texture, Sampler, Shader

Functions

HLSL, Assembly

Effect File Parameters

Technique T0

Global Variables

Effect States

Pass P0

...

The **effects framework** allows effect developers to group similar graphics techniques in a single file. It also allows the content creators to create a single set of content that will work across multiple PC configurations while maintaining a consistent look and feel. The effects framework allows techniques to range from high-level shader, low-level shader, and fixed-function pipeline.

Here are some features of **effects**:

- The **Global Variables** of effects can be set by either the effect itself or by the application.
- The **State Management** of effects includes pipeline, texture, sampler, shader state management. The pipeline state management includes setting transformations, lighting, materials, and rendering options. The texture and sampler state management contains specifying texture files, initializing texture stages, creating sampler objects, and setting sampler state. The shader state management includes creating and deleting shader, setting shader constants, setting shader state, and rendering with shader.
- The **Effects** contain multiple rendering options called **Techniques**. Each technique encapsulates global variables, pipeline state, texture and sampler state, and shader state. A single style is implemented in a rendering pass.

One or more passes can be encapsulated in a technique. All of the passes and techniques can be validated to see if the effect code will run on the hardware device. Effects will save and restore state, leaving the device in the same state as before the effect was run.

- The ***Function*** is defined by HLSL or Assembly program.

Effect Parameters

Effect parameters are all the *non-static variables* declared in an effect. This can include global variables and annotations. Effect parameters can be shared between different effects by declaring parameters with the shared keyword and then creating the effect with an effect pool. A *global variable* with the static keyword in front is not an effect parameter.

Annotations provide a mechanism for adding user information to effect parameters. It is used to exchange data between an application and a GPU program. Annotation declarations are delimited by angle brackets, <>. For example,

```
float3 LightVector < string UILightVector = "Light Vector"; > = {1.0, -1.0, 0.0};
```

Usually an annotation starts with a data type (string), followed by an assign statement with variable name (UILightVector) and the value ("Light Vector"), and ended by a semicolon. The annotation is declared behind a variable and in front of the assignment of the initial value {1.0, -1.0, 0.0}.

Effect States

Effects simplify managing pipeline state. States of effects specify the render conditions of the pipeline. Effects render a given technique that may contain one or more passes. State is restored each time a technique finished. However there are no state changes between passes. Effect states can be divided into the following functional areas:

Pipeline States include light states, material states, render states, vertex render states, and pixel render states.

Sampler States include sampler states and sampler stage states

Texture States include texture states and texture stage states

Shader States include shader constant states, vertex shader constant states, and pixel shader constant states

Finally there are Transform States.

Notes: State description in details can be found in the DirectX SDK documentation.

Effects in Adputer Composer

Adputer Composer can complete real-time video post-processing through programmable GPU. Users can apply image-processing techniques to video and implement these techniques with HLSL in GPU to achieve 10x performances than CPU in computing.

In GPU a video frame is treated as a texture and is rendered over a scene, then the final result is processed through pixel shader. The pos-processing effects can achieve interesting results such as picture blur, bloom of the bright spots, or depth of field (DOF) that blurs a pixel based on its depth.

An effect file named as *VideoMixerDev4.fx* implemented all the video post-process by GPU in Adputer Composer. The .fx is the source file of fx effects, .fxo file is the compiled .fx file. More than one techniques that may have one or more passes have been defined in the .fxo file.

Declaration

In the effect file, there are four texture objects with samplers from which the post-process pixel shader will sample. They are defined as textures.

```
#define          MAXNUM_CHANNEL    4

texture         SamplerTex0;
texture         SamplerTex1;
texture         SamplerTex2;
texture         SamplerTex3;
```

sampler source and scene for color, normal, position, and velocity

```
sampler g_Sampler[MAXNUM_CHANNEL] = {
    sampler_state
    {
        Texture       = <SamplerTex0>;
        MinFilter     = Linear;
        MagFilter     = Linear;
        MipFilter     = None;
        AddressU      = Clamp;
        AddressV      = Clamp;
    },
    . . .
}
```

Usually the post-process pixel shader will need to sample a source texture multiple times for a single destination pixel, each at a different location near the

destination pixel. When this is the case, the effect file declares a kernel which is an array of texel offsets (float2) from the destination pixel. The effect file then defines another equally-sized array of float2. For example,

```
float2 PixelKernel[KernelSize] = { { -1, 0 }, { 0, 0 }, { +1, 0 } };  
float2 TexelKernel[KernelSize]  
<  
    string ConvertPixelsToTexels = "PixelKernel";  
>;
```

This second array has a string annotation, named `ConvertPixelsToTexels`, which contains the name of the first kernel array. When the sample loads the effect file and sees this annotation, it will translate the offsets from pixel coordinates to texel coordinates, which are compatible with the texture sampling functions such as `tex2D`. When the source texels are sampled, the offsets in the kernel are added to the texture coordinates to obtain the texel values near the destination pixel.

Parameters

Some effects use parameters that the user can adjust. Each `PostProcess` technique can define one or more annotations for this purpose. The maximum number of parameters supported by a single technique is four.

Below is the annotation name list:

`Parameter0` type is a string that contains the name of the variable that the user can adjust.

`Parameter0Def` type is a float4. It is the default value for `Parameter0`. If fewer than four floats are needed, use `Parameter0Size` to specify it

`Parameter0Size` type is `Int`. It specifies how many components of `Parameter0` are used. For example if the value is 2, then only the first two components (x and y) are used.

`Parameter0Desc` type is string. It describes this parameter. For example, pass `p0 < float fScaleX = 4.0f; float fScaleY = 4.0f; >`

Here is the vertex input shader structure

```
struct VS_INPUT {
    float4 Position : POSITION;
    float2 TexCoord : TEXCOORD0;
};
```

Here is the vertex output shader structure

```
struct VS_OUTPUT {
    float4 Position : POSITION;
    float2 TexCoord0 : TEXCOORD0;
    float2 TexCoord1 : TEXCOORD1;
    float2 TexCoord2 : TEXCOORD2;
    float2 TexCoord3 : TEXCOORD3;
};
```

Timer is a time variable.

```
float Timer : TIME < string UIWidget = "None"; >;
```

```
//*****
```

```
// Name          VertScene
// Description    vertex shader
// Parameters     inputs Pos, Tex0, Tex1, Tex2, Tex3,
//                outputs oPos, oTex0, oTex1, oTex2, oTex3
// Return        void
```

```
//*****
```

```
void VertScene(
```

input textures

```
    in float4 Pos : POSITION,
    in float2 Tex0 : TEXCOORD0,
    in float2 Tex1 : TEXCOORD0,
    in float2 Tex2 : TEXCOORD0,
    in float2 Tex3 : TEXCOORD0,
```

output textures

```
    out float4 oPos : POSITION,
    out float2 oTex0 : TEXCOORD0,
    out float2 oTex1 : TEXCOORD1,
    out float2 oTex2 : TEXCOORD2,
    out float2 oTex3 : TEXCOORD3)
```

```
{
```

set position and textures

```

    oPos = mul( Pos, WorldViewProj );
    oTex0 = Tex0;
    oTex1 = Tex1;
    oTex2 = Tex2;
    oTex3 = Tex3;
}

```

```

//*****
// Name          PixScene
// Description    Pixel shader
// Parameters     input float2 array Tex[] with TEXCOORD0
// Return        float4 with COLOR0
//*****
float4 PixScene(in float2 Tex[MAXNUM_CHANNEL]:TEXCOORD0) : COLOR0
{
    float4 OUT;

```

get samplers color0 and color1

```

    float4 color0 = tex2D(g_Sampler[0], Tex[0]);
    float4 color1 = tex2D(g_Sampler[1], Tex[1]);

```

get time's sin value to fade two frames

```

    saturate((sin( Timer/2.0 )+1.0)/2.0);
    alpha = 0.5;
    OUT = lerp(color0, color1, alpha) ;
    return OUT;
}

```

```

//*****
// Name          SoloChannel
// Description    Display chan and Tex
// Parameters     chan is the channel number
//               Tex[] is the texture array
// Return        float4
//*****
float4 SoloChannel(in int chan,
                  in float2 Tex[MAXNUM_CHANNEL]:TEXCOORD0)
{
    return tex2D( g_Sampler[chan], Tex[chan]);
}

```

```

//*****
// Name          SoloChannel_1
// Description    Display channel 1 only
// Parameters     Tex[] is the texture array
// Return        float4
//*****
float4 SoloChannel_1(in float2 Tex[MAXNUM_CHANNEL]:TEXCOORD0)
    :COLOR0
{
    return SoloChannel( 0, Tex );
}

//*****
// Name          SoloChannel_1
// Description    Technique SoloChannel_1 transform
// Return        none
//*****
Technique SoloChannel_1
{
    Pass P0
    {

ignore CullMode and ZEnable

        CullMode = None;
        ZEnable = False;

compile VertScene and ScoloChannel_1 by Shader Mode 2.0

        VertexShader = compile vs_2_0 VertScene();
        PixelShader = compile ps_2_0 SoloChannel_1();
    }
}

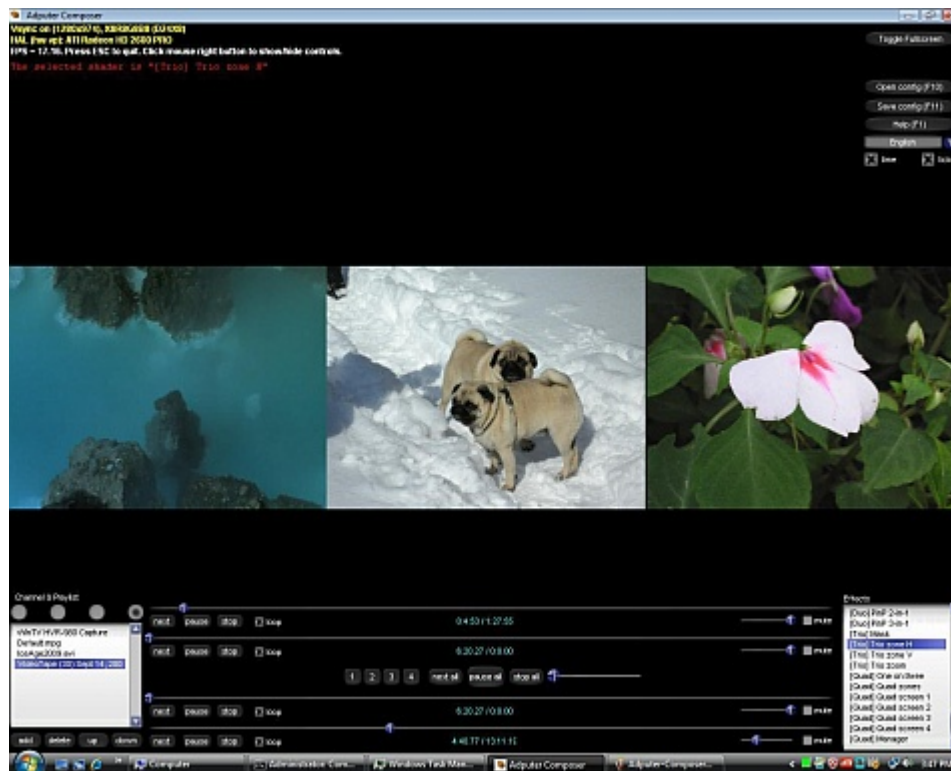
```

Chapter 4 Applications

Digital Signage

Digital Signage is a technology that replaces traditional signs such as posters by large screen liquid crystal displays (LCDs) and plasma display panels (PDPs) as well as embedded computer systems.

The digital signage system can deliver motion pictures and crystal clean logos to capture consumers' attention instantly. Moreover, advertising can be upgraded, distributed, and expressed effectively through digital signage network.



While other media players can only display single video stream, **Adputer Composer** is able to show multiple cinematic HD video streams with smooth mixing and zoning. So content creation is simplified and ads become more attractive with special visual effects.

Digital Signage is widely used in retail stores, banks, restaurants, airports, hotels, sport venues, and apartments.

To make a simple standalone **Digital Signage**, you can do

1. Make a video

- shot a video
- cut video with an creator such as Movie Maker 2
- add audio
- save video

2. Make a logo sign

- design a logo
- place logo onto a black picture with the same size as video
- save picture

3. Design a shader

- select a sample HLSL pixel shader such as circle wipe
- double click to open the source file
- modify parameters for your purpose
- save shader

4. Watch effects

- open video in channel 0
- open logo picture in channel
- select shader in the pixel shader
- launch and watch result

5. Testing

- repeat to achieve best result