Spout 2 Dll

A C compatible library for texture sharing between applications.

spout.zeal.co

Reference Manual

Version 1.00

August 2015

Revisions

Version 1.00 - August 2015

- Initial release.

1. Introduction

The alternative Spout SDK C++ dll project allows the entire Spout SDK to be compiled as a dll.

This has the advantage that all the functions in the Spout SDK classes are available, and the application code is exactly the same as if the SDK files were included in the project.

However, the dll is only suitable for use with C++ projects built using Visual Studio compilers. Other compilers may fail due to the use of "set" functions which they may not support. There is also an issue of "name mangling" that can arise.

Therefore, this separate dll project has been created to provide C compatible exported functions.

The dll itself must be built using Visual Studio. But, once compiled, the dll is compatible with compilers other than Microsoft Visual Studio.

The resulting dll is simple, but there can only be one sender or receiver in any one application.

The functionality is very similar to that of the Spout 1 developer dll and may be a easy way for conversion of projects that use the Spout 1 dll.

An Openframeworks example is included to test the dll using CodeBlocks and the MingW compiler.

The documentation here is a summary of functions. For more detail, refer to the Spout SDK documentation.

2. Using the dll

The most simple process is the same as for the Spout 1 dll :

For a sender.

1) Setup - Initialize a sender (CreateSender)
2) Draw - Send a texture (SendTexture)
3) Close - Release the sender (ReleaseSender)

For a receiver.

1) Setup - Initialize a receiver (CreateReceiver)
2) Draw - receive a texture (ReceiveTexture) and draw it
3) Close - Release the receiver (ReleaseReceiver)

2.1 Sender functions

CreateSender

```
bool CreateSender  (char *Sendername,
                    unsigned int width,
                    unsigned int height,
                    DWORD dwFormat = 0);
```

Creates a Spout sender. This initializes a DirectX device, creates a shared DirectX texture and links an OpenGL texture to it.

For success

A sender with the name provided is initialized and registered as a Spout sender for all receivers to detect. Now you will need to create a local OpenGL texture that can be used for sending.

For failure

An error has occurred and no further operations can be made.

NOTE: for this dll, you cannot create more than one sender within the same application.

SendTexture

```
bool SendTexture (GLuint TextureID,
                  GLuint TextureTarget,
                  unsigned int width,
                  unsigned int height,
                  bool bInvert=true,
                  GLuint HostFBO = 0);
```

Creates a shared texture for all receivers to access.

The invert flag is optional and by default true. This flips the texture in the Y axis which is necessary because DirectX and OpenGL textures are opposite in Y. If it is set to false no flip occurs and the result may appear upside down.

The host fbo argument is optional, default 0. If an fbo is currently bound and it's ID passed, then that binding is restored within the SendTexture function.

SendImage

```
bool SendImage(unsigned char* pixels,
               unsigned int width,
               unsigned int height,
               GLenum glFormat = GL_RGBA,
               bool bAlignment = true,
               bool bInvert=true);
```

Creates a shared texture using image pixels as the source instead of an OpenGL texture.

The format of the image to be sent is RGBA by default but can be a different OpenGL format, for example GL_RGB or GL_BGRA_EXT.

The *bAlignment* flag can be set to false, which disables the default 4-byte alignment assumed by OpenGL. This may be necessary if, for example the pixels you wish to send are RGB and packed with single alignment.

As for SendTexture, the invert flag is optional and by default true.

DrawToSharedTexture

```
bool DrawToSharedTexture(GLuint TextureID, GLuint TextureTarget,
                         unsigned int width, unsigned int height,
                         float max_x = 1.0, float max_y = 1.0,
                         float aspect = 1.0, bool bInvert = true,
                         GLuint HostFBO = 0);
```

Renders an application texture to the shared texture via an FBO.

- width, height are the dimensions of the texture to be drawn.
- max_x, max_y are maximum extents of the texture that will be drawn.
- aspect is the fraction to be used within the draw vertex coordinates.

For example the default vertex coordinates are -1.0 to +1.0 in X and Y and if aspect is passed as a fraction of this (e.g. 0.75), the result is drawn inside this coordinate system.

The FBO used for texture transfer is internal to the function. If an FBO is currently bound when calling this program, the ID should be passed so that the binding can be restored within the function.

UpdateSender

```
bool UpdateSender (char *Sendername,
                   unsigned int width,
                   unsigned int height);
```

If the dimensions of the local texture that you are sending out change, the sender information that receivers access can be updated by this function. This update is also done by *"SendTexture"*, but *"UpdateSender"* does not require a texture to be sent out.

ReleaseSender

```
void ReleaseSender(DWORD dwMsec = 0);
```

Releases a sender created by CreateSender. All OpenGL and DirectX resources for the sender are freed. The dwMsec argument is a debugging aid to introduce a Sleep delay and may be removed in future releases.

2.2 Receiver functions

CreateReceiver

```
bool CreateReceiver(char* Sendername,
                    unsigned int &width,
                    unsigned int &height,
                    bool bUseActive = false);
```

Creates a receiver which must be done before attempting to receive a texture. This initializes a DirectX device, detects a sender, accesses the shared DirectX texture and links an OpenGL texture to it.

The sender name you provide can be the sender that the receiver should connect to. That sender has to be running for it to be used. This is useful if you know the name of the sender you wish to connect to and only want your receiver to start when that sender is running.

If the optional "bUseActive" flag is passed as true, Spout will attempt to find the "active sender" and will connect to that if it is running.

The "active sender" is the one that was started first or the one that the user has selected last by using "SpoutTray", or by using "SpoutPanel" activated from another Spout receiver.

This is useful for an automatic start, where your receiver will detect and connect to any active sender that is running at the time.

Note that the active sender will also be detected if the sender name string is empty, i.e. the first character is 0. *Do not pass NULL as the name*.

For success

Check the sender name, width and height returned and adjust anything required such as a local texture or the window size etc..

For failure

A sender was not found, so just keep calling *CreateReceiver* until a sender is found. The overhead if no senders are running is very low.

NOTE: for this dll you cannot create more than one receiver within the same application.

ReceiveTexture

```
bool ReceiveTexture(char* Sendername,
                    unsigned int &width,
                    unsigned int &height,
                    GLuint TextureID = 0,
                    GLuint TextureTarget = 0,
                    GLuint HostFBO = 0);
```

Once a receiver has been successfully created, you can receive shared textures from senders. The shared texture is internal to Spout.

The shared texture can be received into a local texture if the ID and Target are passed, or the received shared texture can be used directly for graphics operations.

Any changes to sender size are managed within Spout, however if you are receiving to a local texture, the changed width or height are passed back and must be tested to adjust the receiving texture or for for maintaining window dimensions.

ReceiveImage

```
bool ReceiveImage  (char* Sendername,
                    unsigned int &width,
                    unsigned int &height,
                    unsigned char * pixels,
                    GLenum glFormat = GL_RGBA,
                    GLuint HostFBO = 0);
```

Receives a sender shared texture into image pixels. The same sender size changes are passed back as for ReceiveTexture.

GetImageSize

```
bool GetImageSize  (char* Sendername,
                    unsigned int &width,
                    unsigned int &height,
                    bool &bMemoryMode);
```

This function returns the width, height and sharing mode of a sender. The function can be called before a sender or receiver is initialised and without an OpenGL context

The bMemoryMode flag indicates whether a memoryshare sender is running. If so, the system is not texture share compatible.

This dll does not support memoryshare functions.

ReleaseReceiver

```
void ReleaseReceiver();
```

Releases a receiver created by "CreateReceiver". All OpenGL and DirectX resources are freed for the Receiver.

BindSharedTexture

```
bool BindSharedTexture();
```

This function enables the internal Spout shared texture to be bound for OpenGL operations. It is used in exactly the same way as binding an OpenGL texture. This is useful where a local application texture is not used.

NOTE: since this call locks the shared texture, it very important that UnBindSharedTexture is called immediately following the OpenGL operations so that the texture is unlocked for other functions.

UnBindSharedTexture

```
bool UnBindSharedTexture();
```

Unbinds the the internal Spout shared texture.

DrawSharedTexture

```
bool DrawSharedTexture(float max_x = 1.0,
                       float max_y = 1.0,
                       float aspect = 1.0,
                       bool bInvert = true);
```

Performs a draw using the Spout internal shared texture. This may not be suitable in some environments such as Cinder or openframeWorks but can work in others such as FreeFrameGL.

Optionally "aspect" allows the ratio of width to height to be adjusted, typically this value would be width/height.

GetSenderCount

```
int  GetSenderCount();
```

Returns the number of named Spout texture senders running.

GetSenderInfo

```
bool GetSenderInfo (char* Sendername,
                    unsigned int &width,
                    unsigned int &height,
                    HANDLE &dxShareHandle,
                    DWORD &dwFormat);
```

Returns information of a running sender with a given name which may have been retrieved with "GetSenderName".

The format returned is either a DirectX 11 format or zero for a DirectX 9 sender.

GetActiveSender

```
bool GetActiveSender(char* Sendername);
```

Returns the name of the active sender.

SetActiveSender

```
bool SetActiveSender(char* Sendername);
```

Sets the name of the active sender.

SelectSenderPanel

```
bool SelectSenderPanel(char* message = NULL);
```

Activates an executable program "SpoutPanel.exe" which displays a list of Spout senders and allows the user to select one.

NOTE: SpoutPanel.exe must be in the same folder as the host program.

The details of the sender size and DirectX texture format are shown in the dialog.

If a "*/DX9*" argument is passed, the dialog shows a warning if the DirectX format is incompatible and the selection of a DirectX texture is disabled. This is useful if your application is using DirectX 9 functions and you wish to ensure that compatible senders are selected.

An optional text argument can be passed to this function, and then the dialog has a function equivalent to a Windows MessageBox. This is useful when the host application may freeze or is otherwise interrupted by a typical modal MessageBox.

2.3 Utility

SetDX9

```
void SetDX9(bool bDX9 = true);
```

Sets whether the program will operate with DirectX 9 textures.

GetDX9

```
bool GetDX9();
```

Returns whether the the program is operating with DirectX9 textures.

SetDX9compatible

```
void SetDX9compatible(bool bCompatible = true);
```

Sets whether the format of the shared DirectX11 texture that is linked by way of the OpenGL/DirectX interop is compatible with DirectX 9 receivers.

GetDX9compatible

```
bool GetDX9compatible();
```

Returns whether the the format of the shared DirectX11 texture is set to be compatible with DirectX 9 receivers.

SetVerticalSync

```
bool SetVerticalSync(bool bSync = true);
```

Calls *wglSetSwapIntervalEXT() to s*et the OpenGL colour buffer swap periodicity to one video frame (true), so that the draw cycle is locked to to monitor vertical sync, or zero (false) so that buffer swaps are not synchronized to a video frame.

GetVerticalSync

```
int GetVerticalSync();
```

Retrieves the sync state state using *wglGetSwapIntervalEXT();*

NOTE: Vertical sync is affected by many factors. Also applications may manage this already. These functions may be removed in future revisions.

2.4 OpenGL context

An OpenGL context must be established before any of the Spout initialisation or texture sharing functions can be called.

There can be situations where the application does not use OpenGL, such as for DirectX or pixel-based imaging applications.

In these cases, an OpenGL context can be created so that Spout functions can be used.

```
bool InitOpenGL();
bool CloseOpenGL();
```

Do not forget to close any OpenGL context created before exit