Spout SDK

A software development kit for texture sharing between applications.

spout.zeal.co

Reference Manual

Version 1.0

Alpha testing

1. Introduction

Spout SDK is a set of C++ classes that allow programmers to compile Spout texture sharing functions into their own application.

An OpenGL context must be established before any of these functions can be called.

Using the classes

Add all the class files to your project.

spout.h
spoutSDK.h
spoutSDK.cpp
spoutSender.h
spoutSender.cpp
spoutReceiver.h
spoutReceiver.cpp
spoutSenderNames.h
spoutSenderNames.cpp
spoutGLDXinterop.h
spoutGLDXinterop.cpp
spoutGLextensions.cpp
spoutGLextensions.h
spoutMemoryshare.cpp
spoutMemoryshare.h
spoutDirectX.h
spoutDirectX.cpp

In you header file declare :

```
#include "spout.h"
```

It is a good idea to put the Spout class files in a separate folder such as "SpoutSDK", perhaps a level or two above your application project.

For example this might be :

```
#include "..\..\SpoutSDK\spout.h"
```

2. Using Spout

To use Spout, create receiver and sender objects and use them to perform the functions available for sender or receiver.

This can be done in two ways. For example :

```
SpoutSender mySender
```

or

```
SpoutSender * mySender
mySender = new SpoutSender;
. . .
delete mySender;
mySender = NULL;
```

The first method is simple and the objects are deleted and all resources freed when the program terminates.

The second method is useful where you want to create and delete Spout objects while your program is running. In this case you have to specifically delete the objects before the program terminates.

2.1 Creating senders and receivers

For a sender :

1) Create a SpoutSender object

```
SpoutSender mySender;
```

2) Create a sender

```
mySender.CreateSender(SenderName, Width, Height);
```

3) Send a texture

```
mySender.SendTexture(myTextureID, myTextureTarget,
                     Width, Height);
```

For a receiver :

1) Create a SpoutReceiver object

```
SpoutReceiver myReceiver;
```

2) Create a receiver

```
myReceiver.CreateReceiver(SenderName, Width, Height);
```

3) Receive a texture

```
myReceiver.ReceiveTexture(SenderName, Width, Height);
```

4) Draw the texture

```
myReceiver.DrawSharedTexture();
```

Other options are available to bind and unbind the shared texture as may be required or to receive a texture into a local texture.

To bind and unbind the shared texture

```
myReceiver.BindSharedTexture();

myReceiver.UnBindSharedTexture();
```

To receive a texture into a local texture

```
myReceiver.ReceiveTexture(SenderName,
                          width, height,
                          myTextureID,
                          myTextureTarget);
```

2.2 Releasing senders and receivers

You may wish to create Spout senders or receivers, release them and create them again within your program. This may be necessary if, for example, the OpenGL context is lost or changes when initializing something or creating a new window and the like.

The Spout objects do not need to be closed, but the sender or receiver should be released and re-created.

```
mySender.ReleaseSender();
```

or

```
myReceiver.ReleaseReceiver();
```

Once released, the sender or receiver have to be created again before any of the functions will work. The objects themselves remain and can be re-used to re-create a sender or receiver.

2.3 Creating a sender

You must create a sender by calling *CreateSender* before attempting to send a texture. This initializes a DirectX 11 device, creates a shared DirectX texture and links an OpenGL texture to it.

```
mySender.CreateSender(SenderName, width, height);
```

For success

A sender with the name provided has been 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 for this sender object.

NOTE: you cannot create more than one sender for the one sender object. If you require more than one sender, you have to create more sender objects.

2.4 Sending a texture

Once a sender has been successfully created, you can send textures with it.

To send a texture :

First, fill a local texture with the data you want to send. For example :

```
// Grab the screen into a local texture
glBindTexture(GL_TEXTURE_2D, myTexture);
glCopyTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, 0, 0, width, height);
glBindTexture(GL_TEXTURE_2D, 0);
```

Then send it out. This creates a shared texture for all receivers to access.

```
mySender.SendTexture(myTexture, GL_TEXTURE_2D, width, height);
```

NOTE : If you call SendTexture with a framebuffer object bound, that binding will be lost and must be restored afterwards because Spout makes use of its own FBO for intermediate rendering.

If the dimensions of the local texture that you are sending out change, the sender is updated and re-initialized within the SendTexture function.

However, if you wish to update your sender without having to send a texture out you can do so with this function.

```
mySender.UpdateSender(SenderName, width, height);
```

2.5 Releasing a sender

A sender can be released so that all resources are freed. This releases the GL/DX interop, DirectX and the linked DirectX and OpenGL textures. Then a new sender can be created when required.

```
mySender.ReleaseSender();
```

When a sender is released, the sender's name is removed from the list of Spout senders and no receivers will detect it again.

The SpoutSender object cannot be used again to send a texture until you create a new sender.

It is good practice to release the sender at program termination, although all resources are freed even if you do not.

2.6 Creating a receiver

You should create a receiver by calling *CreateReceiver* before attempting to receive a texture. This initializes a DirectX 11 device, detects a sender, accesses the shared DirectX texture and links an OpenGL texture to it.

```
myReceiver.CreateReceiver(SenderName, width, height);
```

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 you provide a NULL sender name, 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 by 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.

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: you cannot create more than one receiver for the one receiver object. If you require more than one receiver, you have to create more receiver objects.

2.7 Receiving a texture

Once a receiver has been successfully created, you can receive textures with it.

NOTE: It is also possible to use ReceiveTexture directly without first creating a Receiver. The same sender name, width and height are returned when a sender is found.

To receive a shared texture from a sender

```
myReceiver.ReceiveTexture(SenderName, Width, Height);
```

The received shared texture can be used directly, for example for graphics operations :

```
glEnable(GL_TEXTURE_2D);
myReceiver.BindSharedTexture();
glBegin(GL_QUADS);
glTexCoord2f(0.0, 1.0);   glVertex2f(-1.0,-1.0);
glTexCoord2f(0.0, 0.0);   glVertex2f(-1.0, 1.0);
glTexCoord2f(1.0, 0.0);   glVertex2f( 1.0, 1.0);
glTexCoord2f(1.0, 1.0);   glVertex2f( 1.0,-1.0);
glEnd();
myReceiver.UnBindSharedTexture();
glDisable(GL_TEXTURE_2D);
```

or a simple quad draw :

```
myReceiver.DrawSharedTexture();
```

To receive an application texture from a sender

```
myReceiver.ReceiveTexture(SenderName,
                          width, height,
                          myTextureID,
                          myTextureTarget);
```

If you are using a local texture and it was received successfully, the width or height could have changed if the sender changed dimensions, so this always has to be tested for maintaining the correct window dimensions or to adjust the receiving texture.

If you have received into a local texture, that can be used as required.

2.8 Releasing a receiver

A receiver can be released so that all resources are freed. This releases the GL/DX interop, DirectX and the linked DirectX and OpenGL textures. Then a new receiver can be created when necessary.

```
myReceiver.ReleaseReceiver();
```

After a receiver is released, you can create a new receiver before receiving textures once more.

It is good practice to release the receiver at program termination, although, for the current version, all resources are freed even if you do not. This will ensure compatibility with future revisions.

2.9 User sender selection

SelectSenderPanel

```
receiver.SelectSenderPanel();
```

This receiver function activates an executable program *"SpoutPanel.exe"* that displays a dialog with a list of names for the currently running senders and allows the user to choose one.

SpoutPanel.exe has to be in the same folder as the host executable program.

Once the user has selected a new sender, *ReceiveTexture* will detect the change and return a different sender name, width and height.

The width and height have to be tested within your program so that the render window or local texture size can be adjusted.

The sender name is not important unless you wish to display which sender is currently being received.

2.10 Creating your own sender list

If you want to create a list of Spout senders in a menu or drop-down list, you can use the following receiver functions.

Getting the sender count

```
int nSenders;
nSenders = myReceiver.GetSenderCount();
```

GetSenderCount returns the number of Spout senders that are currently running.

Getting a sender name

GetSenderName returns the name of a sender with an index within the range indicated by GetSenderCount.

The maximum length of a Spout sender name is 256 characters, but you can specify a character array for the name with a length less than this for *GetSenderName*. The default is 256 if you do not specify a length.

```
char SenderName[64];
int MaxSize = 64;
int index = 0; // or based on GetSenderCount
myReceiver.GetSenderName(index, SenderName, MaxSixe);
```

Getting sender details

GetSenderInfo returns the width, height of the sender as well as the share handle and format of the DirectX 11 shared texture. For OpenGL you will only need to know the width and height.

```
unsigned int width, height;
HANDLE hShareHandle;
DWORD dwFormat;

myReceiver.GetSenderInfo(SenderName,
                         width, height,
                         hShareHandle, dwFormat);
```

If, for example, you want to construct a menu or list :

```
int index, nSenders;
char SenderName[64];
int MaxSize = 64;
 unsigned int width, height;
HANDLE hShareHandle;
DWORD dwFormat;

nSenders = myReceiver.GetSenderCount();
if(nSenders > 0) {
    for(index=0; index<nSenders; index++) {
        myReceiver.GetSenderName(index, SenderName, MaxSize);
        myReceiver.GetSenderInfo(SenderName,
                                 width, height,
                                 hShareHandle, dwFormat);
        //
        // Do something to create your list
        //

    }
}
```

Setting the active sender

When the user has selected a sender from your list, you may wish to set this sender as "active" in keeping with the function of *"SelectSenderPanel"*.

```
myReceiver.SetActiveSender(SenderName);
```

Once set, this sender will be the one that a receiver detects when it starts if the nominated sender name is not found.

Finding the active sender

When you make your selection list, you may wish to indicate to the user which sender is currently "active".

```
char SenderName[256];
myReceiver.GetActiveSender(SenderName);
```

This will return the name of the active sender. Make sure you have prepared a receiving character array for the sender name of no less than 256 bytes in length.

2.11 Finding sender size and compatibility

```
myReceiver.GetImageSize(Sendername, width, height, bMemoryMode);
```

It is sometimes useful to know whether a sender that is running has initialized in Memory Share or Texture Share mode and what the size of that sender is.

The significance of this function is that it does not require an OpenGL context and can be used when a context is not available.

If you find that a sender is running in MemoryShare mode, you can elect to set MemoryShare mode for our application using :

```
myReceiver.SetMemoryShareMode(true);
```

Thereafter, all calls to ReceiveTexture or SendTexture will operate in Memory Share mode. You can also use ReceiveImage and SendImage to transfer data from image pixels directly by way of the CPU.

2.12 Utility functions

Utility functions are available for both sender and receiver objects.

Texture sharing compatibility

The Spout initialization functions *CreateSender* and *CreateReceiver* will detect whether the hardware is compatible with DirectX texture sharing.

If the hardware is not compatible, Spout will default to sharing textures by way of shared memory.

For shared memory, the functions for sending and receiving are exactly the same, but there can only be one sender and one receiver and both must be operating in memory share mode. Multiple senders cannot be detected or used.

You may wish to determine this capability at the beginning of your program so that you can act accordingly.

```
bool bMemoryMode = myReceiver.GetMemoryShareMode();
```

This returns true if memory sharing will be used because the hardware is not texture share compatible.

Setting memory sharing mode

If you want your program to use memory sharing specifically, you can set this at the beginning.

```
mySender.SetMemoryShareMode(true); // default is false
```

Setting DirectX mode

Spout can be initialized to use either DirectX 9 or DirectX 11 textures. You may wish to use DirectX 9 if there are hardware or software compatibility problems. You must set this option at the beginning of the program.

```
mySender.SetDX9(true); // default is false (DirectX 11)
```

Selecting a DirectX 11 sender texture format

If you are using DirectX 11, the texture formats for sharing between DirectX11 and DirectX 9 are limited. DirectX 11 texture format is set by default to (`DXGI_FORMAT_B8G8R8A8_UNORM`) which is compatible with DirectX 9 receivers.

The alternative format is (`DXGI_FORMAT_R8B8G8A8_UNORM`) which is not compatible with DirectX 9 but may be necessary for your application. In this case, only DirectX 11 receivers will pick it up. You must set this at the beginning of the program.

```
mySender.SetDX9compatible(false); // default is true
```

3. Spout functions

3.1 Sender functions

3.1.1 CreateSender

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

Creates a Spout sender. This initializes a DirectX 11 device, creates a shared DirectX texture and links an OpenGL texture to it.

For success

A sender with the name provided has been 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 for this sender object.

NOTE: you cannot create more than one sender for the one sender object. If you require more than one sender, you have to create more sender objects.

3.1.2 SendTexture

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

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.

3.1.3 SendImage

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

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

3.1.4 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 is updated and re-initialized within the SendTexture function.

However, if you wish to update your sender without having to send a texture out you can do so with this function.

3.1.5 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 will be removed in the final release.

3.1.6 SetMemoryShareMode

```
bool SetMemoryShareMode(bool bMemoryMode = true);
```

Sets Spout to use shared memory image transfer instead of the OpenGL/DirectX interop. There can be only one sender/receiver pair which are un-named and the sender selection is no applicable.

3.1.7 GetMemoryShareMode

```
bool GetMemoryShareMode();
```

Returns the current MemoryShare mode. This may have been set by CreateSender if the hardware is not texture share compatible.

3.1.8 SetDX9

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

Sets whether the program will operate with DirectX 9 textures.

3.1.9 GetDX9

```
bool GetDX9();
```

Returns whether the the program is operating with DirectX9 textures.

3.1.10 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.

3.1.11 GetDX9compatible

```
bool GetDX9compatible();
```

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

3.2 Receiver functions

3.2.1 CreateReceiver

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

Creates a receiver which must be done before attempting to receive a texture. This initializes a DirectX 11 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 you provide a NULL sender name, 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.

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: you cannot create more than one receiver for the one receiver object. If you require more than one receiver, you have to create more receiver objects.

3.2.2 ReceiveTexture

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

3.2.3 ReceiveImage

```
bool ReceiveImage  (char* Sendername,
                    unsigned int &width,
                    unsigned int &height,
                    unsigned char * pixels,
                    GLenum glFormat = GL_RGB);
```

Receives a sender shared texture into image pixels. The same sender size changes are passed back as for RecieveTexture. The OpenGL format for the texture transfer is not changeable at this stage but may be for future releases. Receiving images must be RGB.

3.2.4 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 without an OpenGL context

This can be used for detection of a memory share sender if a named texture share sender is not running. This is useful at the start of a program to determine whether to receive from a memoryshare sender.

In that case "*SetMemoryShareMode*" is used as detailed below and subsequent Spout functions will operate in memoryshare mode.

3.2.5 ReleaseReceiver

```
void ReleaseReceiver();
```

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

3.2.6 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.

Since this call locks the shared texture, it very important that UnBindSharedTexture is called immediately following the OpenGL operations.

3.2.7 UnBindSharedTexture

```
bool UnBindSharedTexture();
```

Unbinds the the internal Spout shared texture.

3.2.8 DrawSharedTexture

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

Performs a quad 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.

3.2.9 GetSenderCount

```
int  GetSenderCount();
```

Returns the number of named Spout texture senders running.

3.2.10 GetSenderName

```
bool GetSenderName(int index, char* Sendername, int MaxSize = 256);
```

Returns the name of a sender with an index in the range of GetSenderCount. MaxSize is the maximum number of bytes that returned, up to a maximum of 256 bytes which is the size of a Spout sender name.

NOTE : the name for all other functions that return a Sender name must be dimensioned at least 256 bytes.

3.2.11 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.

3.2.12 GetActiveSender

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

Returns the name of the active sender.

3.2.13 SetActiveSender

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

Sets the name of the active sender.

3.2.14 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 a temporary aid if some Spout applications are using the old Spout dll.

The only DirectX 11 texture format that is compatible with DirectX 9 is DXGI_FORMAT_B8G8R8A8_UNORM (87).

Spout uses DXGI_FORMAT_R8G8B8A8_UNORM (28) as the default format.

An optional text message can be passed to this function, and then the dialog is replaced by a MessageBox with the given text.

This is useful for debugging when the host application may freeze or be otherwise interrupted by a typical modal MessageBox.

The disadvantage is that the MessageBox may move behind a window if the user does not close it immediately and so it is not recommended for users and this option may be removed for release.

3.2.15 Utility

```
bool GetMemoryShareMode();
bool SetMemoryShareMode(bool bMemory = true);
void SetDX9(bool bDX9 = true);
bool GetDX9();
void SetDX9compatible(bool bCompatible = true);
bool GetDX9compatible();
```

These functions are identical to the equivalents for a sender as detailed above.