SpoutDX

For applications not using OpenGL, DirectX textures can be shared with the Spout protocol by using a sub-set of the Spout SDK classes.

Requirements

To share between Spout applications, including those using OpenGL, DirectX devices and textures must comply with the requirements outlined in the NVIDIA *WGL_NV_DX_interop* specifications :

https://www.opengl.org/registry/specs/NV/DX_interop.txt
https://www.opengl.org/registry/specs/NV/DX_interop2.txt

If a D3D11 device is created in your application, the **D3DCREATE_MULTITHREADED** behaviour flag must be set.
The specification only requires that **D3D11_CREATE_DEVICE_SINGLETHREADED** is **not** set, so the flag can be zero.

Shared textures must be created with the following :

1) Description Usage flag - **D3D11_USAGE_DEFAULT**
2) Description MiscFlags - **D3D11_RESOURCE_MISC_SHARED**
3) BindFlags - **D3D11_BIND_RENDER_TARGET |D3D11_BIND_SHADER_RESOURCE**
4) Format **DXGI_FORMAT_B8G8R8A8_UNORM** or **DXGI_FORMAT_R8G8B8A8_UNORM**

Use **DXGI_FORMAT_B8G8R8A8_UNORM** if compatiblity with DirectX 9 is required.

Usage

1) Use methods from the Spout SDK classes directly.
   - A DirectX 11.0 device must be available.

2) Use a support class for simple sending and receiving functions.
   - A DirectX 11.0 device can be created or passed from the application.

Examples are commented throughout. Search “SPOUT” for details. Following is a summary of the steps required.

1. Basic examples

Add these files to your project

```
SpoutSenderNames.h
SpoutSenderNames.cpp
SpoutDirectX.h
SpoutDirectX.cpp
SpoutFrameCount.h
SpoutFrameCount.cpp
SpoutSharedMemory.h
SpoutSharedMemory.cpp
SpoutCopy.h
SpoutCopy.cpp
SpoutUtils.h
SpoutUtils.cpp
```

Sender

Setup

Includes

```
// Change paths as required
#include "SpoutSenderNames.h" // for sender creation and update
#include "SpoutDirectX.h" // for creating a shared texture
#include "SpoutFrameCount.h" // for mutex lock and new frame signal
#include "SpoutUtils.h" // for logging utilites
```

Objects

```
spoutSenderNames spoutSender;
spoutDirectX spoutdx;
spoutFrameCount frame;
```

Global variables (for Tutorial04 basic example)

```
char g_SenderName[256];
unsigned int g_Width = 0;
unsigned int g_Height = 0;
ID3D11Texture2D* g_pSharedTexture = nullptr; // Texture to be shared
HANDLE g_dxShareHandle = NULL; // Share handle for the sender
bool bSpoutInitialized = false;
```

Enable Spout logging if required

```
// OpenSpoutConsole(); // console only for debugging
// EnableSpoutLog(); // Log to console
// EnableSpoutLogFile("Tutorial04.log); // Log to file
// SetSpoutLogLevel(SPOUT_LOG_WARNING); // show only warnings and errors
```

After window and device creation

Give the sender a name

```
strcpy_s(g_SenderName, 256, "Tutorial04");
```

Create a sender mutex for access to the shared texture

```
frame.CreateAccessMutex(g_SenderName);
```

During render

Send the texture (the Tutorial04 example uses the backbuffer).

1. Get the texture details (width height and format)
2. If a sender has not been created yet :
   - Create a shared texture of the same size and format. This should be either DXGI_FORMAT_B8G8R8A8_UNORM or DXGI_FORMAT_R8G8B8A8_UNORM.
   - Create a sender using the shared texture handle
   - Create a sender mutex to control access to the shared texture
   - Enable frame counting for sender frame number and fps
3. If the sender has already been created
   - Check for size change
     - Update the sender and global variables
4. Send the texture
   - Check the sender mutex for texture access
     - Copy the application texture to the sender shared texture
     - Flush immediate context
     - Set a new frame
     - Allow access to the shared texture

Fps control

Hold a target rate if necessary – see code comments

```
frame.HoldFps(60);
```

On program close

```
spoutSender.ReleaseSenderName(g_SenderName);
if (g_pSharedTexture)
	g_pSharedTexture->Release();
```

Receiver

Includes

```
#include "..\..\..\SpoutSDK\SpoutSenderNames.h" // for sender creation and update
#include "..\..\..\SpoutSDK\SpoutDirectX.h" // for creating a shared texture
#include "..\..\..\SpoutSDK\SpoutFrameCount.h" // for mutex lock and new frame signal
#include "..\..\..\SpoutSDK\SpoutUtils.h" // for logging utilites
#include <direct.h> // for _getcwd
#include <TlHelp32.h> // for PROCESSENTRY32
#include <tchar.h> // for _tcsicmp
```

Objects

```
spoutSenderNames spoutSender; // the sender receiving from
spoutDirectX spoutdx;
spoutFrameCount frame;
```

Global variables and utility functions (for Tutorial07 basic example)

```
ID3D11Texture2D* g_pReceivedTexture = nullptr; // Texture received from a sender
ID3D11ShaderResourceView* g_pSpoutTextureRV = nullptr; // Shader resource view
char g_SenderName[256]; // Sender name
char g_SenderNameSetup[256]; // Sender name to connect to
unsigned int g_Width = 0; // Sender width
unsigned int g_Height = 0; // sender height
long g_senderframe = 0; // Sender frame number
double g_senderfps = 0.0; // Sender frame rate
bool bNewFrame = false; // The received frame is new
bool bSpoutInitialized = false; // Initialized for the connected sender
bool bSpoutPanelOpened = false; // User opened sender selection panel
bool bSpoutPanelActive = false; // Selection panel is still open
SHELLEXECUTEINFOA g_ShExecInfo; // Global info so the exit code can be tested
bool OpenSpoutPanel(); // User sender selection dialog
bool CheckSpoutPanel(char *sendername, int maxchars = 256);
```

Enable Spout logging if required

```
// OpenSpoutConsole(); // Console only for debugging
// EnableSpoutLog(); // Log to console
// EnableSpoutLogFile("Tutorial07.log); // Log to file
// SetSpoutLogLevel(SPOUT_LOG_WARNING); // Show only warnings and errors
```

After window and device creation

Optionally set the name of the sender to receive from.
The receiver will only connect to that sender.
The user can over-ride this by selecting another.

```
// strcpy_s(g_SenderNameSetup, 256, "Spout DX11 Sender"); // Set the starting name
// strcpy_s(g_SenderName, 256, "Spout DX11 Sender"); // Set the general name as well
```

During render

1. Create initial width and height variables and set to current global values for testing sender size change
2. Check for user activation of the sender selection dialog
3. If it has been selected, the sender name will be different
   - Reset all variables
   - Close texture access mutex and frame counting for the current sender
   - Reset intialization flag
4. Find if the a sender with the current name exists
   - If a sender was found
     - If not connected yet
       - Create a mutex to control access to the sender shared texture
       - Enable frame counting to get sender frame number and fps
       - Set initialization flag
     - Check for sender size changes
       - Create or re-create the receiving texture
     - Retrieve the sender's shared texture pointer
     - Check for access to the shared texture
       - Test for a new frame from the sender
       - Copy the sender's shared texture to the receiving texture
       - Set a new frame flag (it will be tested later)
       - Allow access to the sender's shared texture
     - For a new frame
       - Use the received texture as required
     - The sender frame number and frame rate can be retrieved
   - If no sender was found
     - If previously connected
       - If a connecting name has been set, reset the sender name to it. Otherwise zero the sender name
       - Zero the sender width and height
       - Close the access mutex and frame counting
       - Clear any application resources using the received texture
5. Continue with render
6. Final swapchain present
7. Hold a target rate if necessary – see code comments

```
frame.HoldFps(60);
```

On program close

Release objects created

For Tutorial07 basic example

```
if (g_pSpoutTextureRV)
        g_pSpoutTextureRV->Release();
if (g_pReceivedTexture)
        g_pReceivedTexture->Release();
```

2. Examples using the SpoutDX support class

Add these files to your project

```
SpoutDX.h / SpoutDX.cpp
SpoutSenderNames.h / SpoutSenderNames.cpp
SpoutDirectX.h / SpoutDirectX.cpp
SpoutFrameCount.h / SpoutFrameCount.cpp
SpoutSharedMemory.h/ SpoutSharedMemory.cpp
SpoutCopy.h / SpoutCopy.cpp
SpoutUtils.h / SpoutUtils.cpp
```

Sender

Setup

Includes

```
// Change paths as required
#include "SpoutDX.h"
```

Objects

```
spoutDX spoutSender;
```

Enable Spout logging if required

```
// OpenSpoutConsole(); // console only for debugging
// EnableSpoutLog(); // Log to console
// EnableSpoutLogFile("Tutorial04.log); // Log to file
// SetSpoutLogLevel(SPOUT_LOG_WARNING); // show only warnings and errors
```

After window and device creation

If a DirectX 11.0 device is available, the device pointer must be passed to the SpoutDX class. Otherwise a device is created within the class and the pointer can be retrieved if necessary with GetDevice().

```
if (!spoutSender.OpenDirectX11(g_pd3dDevice))
        return FALSE;
```

Give the sender a name. If none is specified, the executable name will be used.

```
spoutSender.SetSenderName("Tutorial04sender");
```

During render

Send the texture (the Tutorial04 example uses the backbuffer).
SendTexture handles sender creation and re-sizing

```
spoutSender.SendTexture(pBackBuffer);
```

Hold a target rate if necessary – see code comments

```
spoutSender.HoldFps(60);
```

Receiver

Includes

```
#include "SpoutDX.h"
```

Objects

```
spoutDX spoutReceiver;
```

Global variables (for Tutorial07 example)

```
ID3D11Texture2D* g_pReceivedTexture = nullptr; // Texture received from a sender
// The texture is created after connecting to a sender
ID3D11ShaderResourceView* g_pSpoutTextureRV = nullptr; // Shader resource view
```

Enable Spout logging if required

```
// OpenSpoutConsole(); // Console only for debugging
// EnableSpoutLog(); // Log to console
// EnableSpoutLogFile("Tutorial07.log"); // Log to file
// SetSpoutLogLevel(SPOUT_LOG_WARNING); // Show only warnings and errors
```

Optionally set the name of the sender to receive from. The receiver will only connect to that sender. The user can over-ride this by selecting another.

```
// spoutreceiver.SetReceiverName("Spout DX11 Sender");
```

After window and device creation

If a DirectX 11.0 device is available, the device pointer must be passed to the SpoutDX class. Otherwise a device is created and the pointer can be retrieved if necessary with GetDevice().

```
if (!spoutreceiver.OpenDirectX11(g_pd3dDevice))
        return FALSE;
```

During render

1) Receive a texture from a sender
   - If a sender was found
     - If the sender is updated
       - Create or re-create the receiving texture
     - If the frame is new
       - Create or re-create associated resources
2) If the sender was not found or closed
   - Release the receiving texture
   - Release associated resources

On program close

Close the receiver

```
spoutreceiver.ReleaseReceiver();
```

Release objects created
For Tutorial07 basic example

```
if (g_pSpoutTextureRV)
        g_pSpoutTextureRV->Release();
if (g_pReceivedTexture)
        g_pReceivedTexture->Release();
```