

How to use Stereo API

1. Create Direct3D Device
2. Create StereoAPI

```
    HMODULE      hMod = LoadStereoLibrary();
    IStereoAPI* pAPI;
    CreateStereoAPI( hMod, &pAPI );
```
3. While rendering the scene in your application control resource creation and stereo render mode via API function calls (section **Functions for stereo render control**)
4. Before destroying your application, release Stereo API

```
    pAPI -> Release();
```
5. Destroy Direct3D Device

Three standard COM methods for reference counting

```
HRESULT  QueryInterface( REFIID riid, void** ppvObj )
ULONG    AddRef ( )
ULONG    Release( )
```

API creation functions

```
HMODULE  LoadStereoLibrary( )
```

First, you must load stereo library immediately after creating Direct3D device.

```
HRESULT  CreateStereoAPI( HMODULE hModule, IStereoAPI** pAPI )
```

Creating StereoAPI interface.

hModule - module handle returned from LoadStereoLibrary().

pAPI - address of pointer to IStereoAPI.

You must call Release() before destroying Direct3D device.

Functions for stereo render control

With these functions you may manage of creation and drawing objects while drawing scene.

```
HRESULT  GetDriverVersion( DWORD* pMajor, DWORD* pMinor,
                          DWORD* pBuild, DWORD* pQFE )
```

Getting information about driver version.

pMajor - major version

pMinor - minor version

pBuild - build number

pQFE - internal revision version

```
float    GetSeparation( )
```

Get Stereo Camera Separation value.

```
HRESULT  SetSeparation( float Value )
```

Set Stereo Camera Separation value.

```
Float    GetConvergence( )
```

Get Stereo Camera Convergence value.

```
HRESULT  SetConvergence( float Value )
```

Set Stereo Camera Convergence value.

int GetPreset()

Get current preset index. Preset contains stereo separation and convergence values.

HRESULT SetPreset(**int** Value)

Select another preset.

int GetPresetsCount()

Get total number of presets.

bool IsLaserSightOn()

Return true if laser sight spot is enabled.

HRESULT SetLaserSight(**bool** Value)

Enable or disable laser sight spot.

bool IsAutoFocusOn()

Get autofocus status.

HRESULT SetAutoFocus(**bool** Value)

If enabled, stereo driver select convergence parameter automatically (that parameter will be changed dynamically). Note that active autofocus will increase CPU pressure (it utilizes up to 100% of an idle CPU core if such is available).

bool IsStereoOn()

Return true if stereo render enabled.

HRESULT SetStereoActive(**bool** Value)

Enable render in stereo. All draw calls will be duplicated inside the stereo driver.

bool IsShowOSDOn()

Return true if driver's on screen display (OSD) info enabled.

HRESULT SetShowOSDOn(**bool** Value)

When OSD is enabled some text messages appear in the left top corner of screen. OSD shows various data such as the executable name, render mode and FPS.

bool IsShowFPSOn()

Return true if you wish to show FPS counter via OSD.

HRESULT SetShowFPSOn(**bool** Value)

Show FPS (frames per seconds) in OSD. Note that OSD must be enabled too.

float GetMinSeparation()

Get minimum possible Stereo Camera Separation value.

float GetMaxSeparation()

Get maximum possible Stereo Camera Separation value.

float GetMinConvergence()

Get minimum possible Stereo Camera Convergence value.

float GetMaxConvergence()

Get minimum possible Stereo Camera Separation value.

int GetRenderTargetCreationMode()

Return mode for render targets creation method.

HRESULT SetRenderTargetCreationMode(**int** Value)

A possible parameter values are 0, 1, 2.

- 0 - All render targets always will be created as mono - for ordinal mono render
- 1 - All render targets always will be created as stereo (i.e. duplicated) - for stereo rendering.
- 2 - Driver will decide itself mono or stereo render target to create, using its internal rules.

The following functions define how Draw() calls will be issued. Inside the block all Draw() calls will be stereo or mono, according to the block type. Blocks may be nested. Outside blocks driver will function in its usual mode.

```
HRESULT   BeginMonoBlock   ( )  
HRESULT   EndMonoBlock     ( )  
HRESULT   BeginStereoBlock( )  
HRESULT   EndStereoBlock   ( )
```

```
bool IsLastDrawStereo( )
```

Return true, if last Draw() call was made in stereo, i.e. duplicated to draw left and right.

Functions for stereo blt

These functions allow to handle left and right access, where left and right are on the same surface as opposed to 3D mode. For all these functions:

- 0 - left eye
- 1 - right eye

```
HRESULT   SetBltSrcEye( int Value )
```

Select area in source surface for StretchRect().

```
int   GetBltSrcEye( )
```

Get selected area for source surface for StretchRect().

```
HRESULT   SetBltDstEye( int Value )
```

Select area in destination for StretchRect().

```
Int   GetBltDstEye( )
```

Get selected area for destination surface for StretchRect().

Stereo blt sample

Here is Creation and Paint function that uses stereo blt functions

```
IDirect3DSurface9* g_pLeftSurface;
IDirect3DSurface9* g_pRightSurface;
void CreateSurfaces()
{
    pStereoAPI->SetRenderTargetCreationMode( 0 );
    // create right eye surface or texture
    pD3DDevice->CreateOffscreenPlainSurface( width, height,
Format, pool, &g_pRightSurface, 0 );
    ...
    pStereoAPI->SetRenderTargetCreationMode( 2 );

    ...

    pStereoAPI->SetRenderTargetCreationMode( 0 );
    // create left eye surface or texture
    pD3DDevice->CreateOffscreenPlainSurface( width, height,
Format, pool, &g_pLeftSurface, 0 );
    ...
    pStereoAPI->SetRenderTargetCreationMode( 2 );
}

void Paint()
{
    // preparing right eye surface
    ...
    pStereoAPI->SetBltDstEye( 1 );
    pD3DDevice->StretchRect( g_pRightSurface, srcRect,
pBackbuffer, dstRect, tex_filter );

    ...

    // preparing left eye surface
    ...
    pStereoAPI->SetBltDstEye( 0 );
    pD3DDevice->StretchRect( g_pLeftSurface, srcRect,
pBackbuffer, dstRect, tex_filter );
}
```