Part C

Microsoft® DirectX™ 2 Software Development Kit

C H A P T E R 5

Direct3D Immediate-Mode Reference

Immediate-Mode Reference

Macros

D3DDivide

```
D3DDivide(a, b)    (float)((double) (a) / (double) (b))
```

Divides two values.

- Returns the quotient of the division.

a and *b*
Dividend and divisor in the expression, respectively.

See also **D3DMultiply**

D3DMultiply

```
D3DMultiply(a, b)    ((a) * (b))
```

Multiplies two values.

- Returns the product of the multiplication.

a and *b*
Values to be multiplied.

See also **D3DDivide**

D3DRGB

```
D3DRGB(r, g, b) \
    (0xff000000L | ( ((long)((r) * 255)) << 16) | \
    (((long)((g) * 255)) << 8) | (long)((b) * 255))
```

Initializes a color with the supplied RGB values.

- Returns the **D3DCOLOR** value corresponding to the supplied RGB values.

r, *g*, and *b*
Red, green, and blue components of the color. These should be floating-point values in the range 0 through 1.

See also **D3DRGBA**

D3DRGBA

```
D3DRGBA(r, g, b, a) \
    ((((long)((a) * 255)) << 24) | (((long)((r) * 255)) << 16) |
    (((long)((g) * 255)) << 8) | (long)((b) * 255))
```

Initializes a color with the supplied RGBA values.

- Returns the **D3DCOLOR** value corresponding to the supplied RGBA values.

r, *g*, *b*, and *a*
Red, green, blue, and alpha components of the color.

See also **D3DRGB**

D3DSTATE_OVERRIDE

```
D3DSTATE_OVERRIDE(type) ((DWORD) (type) + D3DSTATE_OVERRIDE_BIAS)
```

Overrides the state of the rasterization, lighting, or transformation module. Applications can use this macro to lock and unlock a state.

- No return value.

type
State to override. This parameter should be one of the members of the **D3DTRANSFORMSTATETYPE**, **D3DLIGHTSTATETYPE**, or **D3DRENDERSTATETYPE** enumerated types.

An application might, for example, use the STATE_DATA macro (defined in the D3dmacs.h header file in the Misc directory of the DirectX 2 SDK sample code) and D3DSTATE_OVERRIDE to lock and unlock the **D3DRENDERSTATE_SHADEMODE** render state:

```
//  Lock the shade mode.

STATE_DATA(D3DSTATE_OVERRIDE(D3DRENDERSTATE_SHADEMODE), TRUE, lpBuffer);

//  Work with the shade mode and unlock it when read-only status is not
required.

STATE_DATA(D3DSTATE_OVERRIDE(D3DRENDERSTATE_SHADEMODE), FALSE, lpBuffer);
```

For more information about overriding rendering states, see **States and State Overrides**.

D3DVAL

```
D3DVAL(val)     ((float)val)
```

Creates a value whose type is **D3DVALUE**.

- Returns the converted value.

val
Value to be converted.

See also **D3DVALP**

D3DVALP

```
D3DVALP(val, prec)    ((float)val)
```

Creates a value of the specified precision.

- Returns the converted value.

val
Value to be converted.

prec
Ignored.

The precision, as implemented by the **D3DVAL** macro, is 16 bits for the fractional part of the value.

See also **D3DVAL**

RGB_GETBLUE

```
RGB_GETBLUE(rgb)    ((rgb) & 0xff)
```

Retrieves the blue component of a **D3DCOLOR** value.

- Returns the blue component.

rgb
Color index from which the blue component is retrieved.

RGB_GETGREEN

```
RGB_GETGREEN(rgb)    (((rgb) >> 8) & 0xff)
```

Retrieves the green component of a **D3DCOLOR** value.

- Returns the green component.

rgb
Color index from which the green component is retrieved.

RGB_GETRED

```
RGB_GETRED(rgb)    (((rgb) >> 16) & 0xff)
```

Retrieves the red component of a **D3DCOLOR** value.

- Returns the red component.

rgb
Color index from which the red component is retrieved.

RGB_MAKE

```
RGB_MAKE(r, g, b)    ((D3DCOLOR) (((r) << 16) | ((g) << 8) | (b)))
```

Creates an RGB color from supplied values.

- Returns the color.

r, *g*, and *b*
Red, green, and blue components of the color to be created. These should be integer values in the range zero through 255.

RGB_TORGBA

```
RGB_TORGBA(rgb)    ((D3DCOLOR) ((rgb) | 0xff000000))
```

Creates an RGBA color from a supplied RGB color.

- Returns the RGBA color.

rgb
RGB color to be converted to an RGBA color.

See also **RGBA_TORGB**

RGBA_GETALPHA

```
RGBA_GETALPHA(rgb)    ((rgb) >> 24)
```

Retrieves the alpha component of an RGBA **D3DCOLOR** value.

- Returns the alpha component.

rgb
Color index from which the alpha component is retrieved.

RGBA_GETBLUE

```
RGB_GETBLUE(rgb)    ((rgb) & 0xff)
```

Retrieves the blue component of an RGBA **D3DCOLOR** value.

- Returns the blue component.

rgb
Color index from which the blue component is retrieved.

RGBA_GETGREEN

```
RGB_GETGREEN(rgb)     (((rgb) >> 8) & 0xff)
```

Retrieves the green component of an RGBA **D3DCOLOR** value.

- Returns the green component.

rgb
Color index from which the green component is retrieved.

RGBA_GETRED

```
RGB_GETRED(rgb)     (((rgb) >> 16) & 0xff)
```

Retrieves the red component of an RGBA **D3DCOLOR** value.

- Returns the red component.

rgb
Color index from which the red component is retrieved.

RGBA_MAKE

```
RGBA_MAKE(r, g, b, a)    \
    ((D3DCOLOR) (((a) << 24) | ((r) << 16) | ((g) << 8) | (b)))
```

Creates an RGBA **D3DCOLOR** value from supplied red, green, blue, and alpha components.

- Returns the color.

r, *g*, *b*, and *a*
Red, green, blue, and alpha components of the RGBA color to be created.

RGBA_SETALPHA

```
RGBA_SETALPHA(rgba, x)     (((x) << 24) | ((rgba) & 0x00ffffff))
```

Sets the alpha component of an RGBA **D3DCOLOR** value.

- Returns the RGBA color whose alpha component has been set.

rgba
RGBA color for which the alpha component will be set.

x
Value of alpha component to be set.

RGBA_TORGB

```
RGBA_TORGB(rgba)    ((D3DCOLOR) ((rgba) & 0xffffff))
```

Creates an RGB **D3DCOLOR** value from a supplied RGBA **D3DCOLOR** value by stripping off the alpha component of the color.

- Returns the RGB color.

rgba
RGBA color to be converted to an RGB color.

See also **RGB_TORGBA**

Callback Functions

D3DENUMDEVICESCALLBACK

```
typedef HRESULT (FAR PASCAL * LPD3DENUMDEVICESCALLBACK)
    (LPGUID lpGuid, LPSTR lpDeviceDescription,
    LPSTR lpDeviceName, LPD3DDEVICEDESC lpD3DHWDeviceDesc,
    LPD3DDEVICEDESC lpD3DHELDeviceDesc, LPVOID lpUserArg);
```

Prototype definition for the callback function to enumerate installed Direct3D devices.

- Applications should return one of the following values:

  D3DENUMRET_CANCEL
  Cancel the enumeration.

  D3DENUMRET_OK
  Continue the enumeration.

lpGuid
Address of a globally unique identifier (GUID).

lpDeviceDescription
Address of a textual description of the device.

lpDeviceName
Address of the device name.

lpD3DHWDeviceDesc
Address containing the hardware capabilities of the Direct3D device.

lpD3DHELDeviceDesc
Address containing the emulated capabilities of the Direct3D device.

lpUserArg
Address of application-defined data passed to this callback function.

When determining the order in which to call callback functions, the system searches the objects highest in the hierarchy first, and then calls their callback functions in the order in which they were created.

D3DENUMTEXTUREFORMATSCALLBACK

```
typedef HRESULT (WINAPI* LPD3DENUMTEXTUREFORMATSCALLBACK)
    (LPDDSURFACEDESC lpDdsd, LPVOID lpUserArg);
```

Prototype definition for the callback function to enumerate texture formats.

lpDdsd
Address of the DirectDrawSurface object containing the texture information.

lpUserArg
Address of application-defined data passed to this callback function.

When determining the order in which to call callback functions, the system searches the objects highest in the hierarchy first, and then calls their callback functions in the order in which they were created.

D3DVALIDATECALLBACK

```
typedef HRESULT (WINAPI* LPD3DVALIDATECALLBACK)
    (LPVOID lpUserArg, DWORD dwOffset);
```

Application-defined callback function supplied when an application calls the **IDirect3DExecuteBuffer::Validate** method. This method is a debugging routine that checks the execute buffer and returns an offset into the buffer when any errors are encountered.

lpUserArg
Address of application-defined data passed to this callback function.

dwOffset
Offset into the execute buffer at which the system found an error.

When determining the order in which to call callback functions, the system searches the objects highest in the hierarchy first, and then calls their callback functions in the order in which they were created.

IDirect3D Interface

IDirect3D Interface Method Groups

Applications use the methods of the **IDirect3D** interface to create Direct3D objects and set up the environment. The methods can be organized into the following groups:

Creation	**CreateLight**
	CreateMaterial
	CreateViewport
Enumeration and initialization	**EnumDevices**
	FindDevice
	Initialize
IUnknown	**AddRef**
	QueryInterface
	Release

All COM interfaces inherit the **IUnknown** interface methods, which are listed in the "IUnknown" group above. These three methods allow additional interfaces to be added to the Direct3D object without affecting the functionality of the original interface.

IDirect3D::AddRef

```
ULONG AddRef();
```

Increases the reference count of the Direct3D object by 1. This method is part of the **IUnknown** interface inherited by Direct3D.

- Returns the new reference count of the object.

When the Direct3D object is created, its reference count is set to 1. Every time an application obtains an interface to the object or calls the **AddRef** method, the object's reference count is increased by 1. Use the **IDirect3D::Release** method to decrease the reference count of the object by 1.

IDirect3D::CreateLight

```
HRESULT CreateLight(LPDIRECT3DLIGHT* lplpDirect3Dlight,
    IUnknown* pUnkOuter);
```

Allocates a Direct3DLight object. This object can then be associated with a viewport by using the **IDirect3DViewport::AddLight** method.

- Returns D3D_OK if successful, or an error otherwise, which may be one of the following values:

  DDERR_INVALIDOBJECT

  DDERR_INVALIDPARAMS

lplpDirect3DLight
Address that will be filled with a pointer to an **IDirect3DLight** interface if the call succeeds.

pUnkOuter
This parameter is provided for future compatibility with COM aggregation features. Currently, however, the **IDirect3D::CreateLight** method returns an error if this parameter is anything but NULL.

IDirect3D::CreateMaterial

```
HRESULT CreateMaterial(LPDIRECT3DMATERIAL* lplpDirect3DMaterial,
    IUnknown* pUnkOuter);
```

Allocates a Direct3DMaterial object.

- Returns D3D_OK if successful, or an error otherwise. For a list of possible return codes, see **Direct3D Immediate-Mode Return Values**.

lplpDirect3DMaterial
Address that will be filled with a pointer to an **IDirect3DMaterial** interface if the call succeeds.

pUnkOuter
This parameter is provided for future compatibility with COM aggregation features. Currently, however, the **IDirect3D::CreateMaterial** method returns an error if this parameter is anything but NULL.

IDirect3D::CreateViewport

```
HRESULT CreateViewport(LPDIRECT3DVIEWPORT* lplpD3DViewport,
    IUnknown* pUnkOuter);
```

Creates a Direct3DViewport object. The viewport is associated with a Direct3DDevice object by using the **IDirect3DDevice::AddViewport** method.

- Returns D3D_OK if successful, or an error otherwise, which may be one of the following values:

  DDERR_INVALIDOBJECT

  DDERR_INVALIDPARAMS

lplpD3DViewport
Address that will be filled with a pointer to an **IDirect3DViewport** interface if the call succeeds.

pUnkOuter
This parameter is provided for future compatibility with COM aggregation features. Currently, however, the **IDirect3D::CreateViewport** method returns an error if this parameter is anything but NULL.

IDirect3D::EnumDevices

```
HRESULT EnumDevices(LPD3DENUMDEVICESCALLBACK lpEnumDevicesCallback,
    LPVOID lpUserArg);
```

Enumerates all Direct3D device drivers installed on the system.

- Returns D3D_OK if successful, or an error otherwise, which may be one of the following values:

  DDERR_INVALIDOBJECT

  DDERR_INVALIDPARAMS

lpEnumDevicesCallback
Address of the **D3DENUMDEVICESCALLBACK** callback function that the enumeration procedure will call every time a match is found.

lpUserArg
Address of application-defined data passed to the callback function.

IDirect3D::FindDevice

```
HRESULT FindDevice(LPD3DFINDDEVICESEARCH lpD3DFDS,
    LPD3DFINDDEVICERESULT lpD3DFDR);
```

Finds a device with specified characteristics and retrieves a description of it.

- Returns D3D_OK if successful, or an error otherwise. For a list of possible return codes, see **Direct3D Immediate-Mode Return Values**.

lpD3DFDS
Address of the **D3DFINDDEVICESEARCH** structure describing the device to be located.

lpD3DFDR
Address of the **D3DFINDDEVICERESULT** structure describing the device if it is found.

IDirect3D::Initialize

```
HRESULT Initialize(REFIID lpREFIID);
```

This method is provided for compliance with the COM protocol.

- Returns **DDERR_ALREADYINITIALIZED** because the Direct3D object is initialized when it is created.

lpREFIID
Address of a universally unique identifier (UUID).

IDirect3D::QueryInterface

```
HRESULT QueryInterface(REFIID riid, LPVOID* obp);
```

Determines if the Direct3D object supports a particular COM interface. If it does, the system increases the reference count for the object, and the application can begin using that interface immediately. This method is part of the **IUnknown** interface inherited by Direct3D.

- Returns D3D_OK if successful, or an error otherwise, which may be one of the following values:

  DDERR_INVALIDOBJECT

  DDERR_INVALIDPARAMS

riid
Reference identifier of the interface being requested.

obp
Address of a pointer that will be filled with the interface pointer if the query is successful.

If the application does not need to use the interface retrieved by a call to this method, it must call the **Release** method for that interface to free it. The **IDirect3D::QueryInterface** method allows Direct3D objects to be extended by Microsoft and third parties without interfering with each other's existing or future functionality.

IDirect3D::Release

```
ULONG Release();
```

Decreases the reference count of the Direct3D object by 1. This method is part of the **IUnknown** interface inherited by Direct3D.

- Returns the new reference count of the object.

The Direct3D object deallocates itself when its reference count reaches 0. Use the **IDirect3D::AddRef** method to increase the reference count of the object by 1.

IDirect3DDevice Interface

IDirect3DDevice Interface Method Groups

Applications use the methods of the **IDirect3DDevice** interface to retrieve and set the capabilities of Direct3D devices. The methods can be organized into the following groups:

Execution	**CreateExecuteBuffer**
	Execute
Information	**EnumTextureFormats**
	GetCaps
	GetDirect3D
	GetPickRecords
	GetStats
IUnknown	**AddRef**
	QueryInterface
	Release
Matrices	**CreateMatrix**
	DeleteMatrix
	GetMatrix
	SetMatrix
Miscellaneous	**Initialize**
	Pick
	SwapTextureHandles
Scenes	**BeginScene**
	EndScene
Viewports	**AddViewport**
	DeleteViewport
	NextViewport

All COM interfaces inherit the **IUnknown** interface methods, which are listed in the "IUnknown" group above. These three methods allow additional interfaces to be added to the Direct3DDevice object without affecting the functionality of the original interface.

The Direct3DDevice object is obtained through the appropriate call to the **IDirect3DDevice::QueryInterface** method from a DirectDrawSurface object that was created as a 3D-capable surface.

IDirect3DDevice::AddRef

```
ULONG AddRef();
```

Increases the reference count of the Direct3DDevice object by 1. This method is part of the **IUnknown** interface inherited by Direct3DDevice.

- Returns the new reference count of the object.

When the Direct3DDevice object is created, its reference count is set to 1. Every time an application obtains an interface to the object or calls the **AddRef** method, the object's reference count is increased by 1. Use the **IDirect3DDevice::Release** method to decrease the reference count of the object by 1.

IDirect3DDevice::AddViewport

```
HRESULT AddViewport(LPDIRECT3DVIEWPORT lpDirect3DViewport);
```

Adds the specified viewport to the list of viewport objects associated with the device.

- Returns D3D_OK if successful, or an error otherwise, which may be one of the following values:

  DDERR_INVALIDOBJECT

  DDERR_INVALIDPARAMS

lpDirect3DViewport
Address of the **IDirect3DViewport** interface that should be associated with this Direct3DDevice object.

IDirect3DDevice::BeginScene

```
HRESULT BeginScene();
```

Begins a scene.

- Returns D3D_OK if successful or an error otherwise.

Applications must call this method before performing any rendering, and must call **IDirect3DDevice::EndScene** when rendering is complete.

See also **IDirect3DDevice::EndScene**

IDirect3DDevice::CreateExecuteBuffer

```
HRESULT CreateExecuteBuffer(LPDIRECT3DEXECUTEBUFFERDESC lpDesc,
    LPDIRECT3DEXECUTEBUFFER* lplpDirect3DExecuteBuffer,
    IUnknown* pUnkOuter);
```

Allocates an execute buffer for a display list. The list may be read by hardware DMA into VRAM for processing. All display primitives in the buffer that have indices to vertices must also have those vertices in the same buffer.

- Returns D3D_OK if successful, or an error otherwise, which may be one of the following values:

  DDERR_INVALIDOBJECT

  DDERR_INVALIDPARAMS

lpDesc
Address of a **D3DEXECUTEBUFFERDESC** structure that describes the Direct3DExecuteBuffer object to be created. The call will fail if a buffer of at least the specified size cannot be created.

lplpDirect3DExecuteBuffer
Address of a pointer that will be filled with the address of the new Direct3DExecuteBuffer object.

pUnkOuter
This parameter is provided for future compatibility with COM aggregation features. Currently, however, this method returns an error if this parameter is anything but NULL.

The **D3DEXECUTEBUFFERDESC** structure describes the execute buffer to be created. At a minimum, the application must specify the size required. If the application specifies DEBCAPS_VIDEO_MEMORY in the capabilities member, Direct3D will attempt to keep the execute buffer in video memory.

The application can use the **IDirect3DExecuteBuffer::Lock** method to request that the memory be moved. When this method returns, it will adjust the contents of the **D3DEXECUTEBUFFERDESC** structure to indicate whether the data resides in system or video memory.

IDirect3DDevice::CreateMatrix

```
HRESULT CreateMatrix(LPD3DMATRIXHANDLE lpD3DMatHandle);
```

Creates a matrix.

- Returns D3D_OK if successful, or an error otherwise, such as **DDERR_INVALIDPARAMS**.

lpD3DMatHandle
Address of a variable that will contain a handle to the matrix that is created. The call will fail if a buffer of at least the size of the matrix cannot be created.

See also **IDirect3DDevice::DeleteMatrix**, **IDirect3DDevice::SetMatrix**

IDirect3DDevice::DeleteMatrix

```
HRESULT DeleteMatrix(D3DMATRIXHANDLE d3dMatHandle);
```

Deletes a matrix handle. This matrix handle must have been created by using the **IDirect3DDevice::CreateMatrix** method.

- Returns D3D_OK if successful, or an error otherwise, such as **DDERR_INVALIDPARAMS**.

d3dMatHandle
Matrix handle to be deleted.

See also **IDirect3DDevice::CreateMatrix**, **IDirect3DDevice::SetMatrix**

IDirect3DDevice::DeleteViewport

```
HRESULT DeleteViewport(LPDIRECT3DVIEWPORT lpDirect3DViewport);
```

Removes the specified viewport from the list of viewport objects associated with the device.

- Returns D3D_OK if successful, or an error otherwise, which may be one of the following values:

  DDERR_INVALIDOBJECT

  DDERR_INVALIDPARAMS

lpDirect3DViewport
Address of the Direct3DViewport object that should be disassociated with this Direct3DDevice object.

IDirect3DDevice::EndScene

```
HRESULT EndScene();
```

Ends a scene that was begun by calling the **IDirect3DDevice::BeginScene** method.

- Returns D3D_OK if successful, or an error otherwise.

See also **IDirect3DDevice::BeginScene**

IDirect3DDevice::EnumTextureFormats

```
HRESULT EnumTextureFormats(
    LPD3DENUMTEXTUREFORMATSCALLBACK lpd3dEnumTextureProc,
    LPVOID lpArg);
```

Queries the current driver for a list of supported texture formats.

- Returns D3D_OK if successful, or an error otherwise, which may be one of the following values:

  DDERR_INVALIDOBJECT

  DDERR_INVALIDPARAMS

lpd3dEnumTextureProc
Address of the **D3DENUMTEXTUREFORMATSCALLBACK** callback function that the enumeration procedure will call for each texture format.

lpArg
Address of application-defined data passed to the callback function.

IDirect3DDevice::Execute

```
HRESULT Execute(LPDIRECT3DEXECUTEBUFFER lpDirect3DExecuteBuffer,
    LPDIRECT3DVIEWPORT lpDirect3DViewport, DWORD dwFlags);
```

Executes a buffer.

- Returns D3D_OK if successful, or an error otherwise, which may be one of the following values:

  DDERR_INVALIDOBJECT

  DDERR_INVALIDPARAMS

lpDirect3DExecuteBuffer
Address of the execute buffer to be executed.

lpDirect3DViewport
Address of the Direct3DViewport object that describes the transformation context into which the execute buffer will be rendered.

dwFlags
Flags specifying whether or not objects in the buffer should be clipped. This parameter must be one of the following values:

D3DEXECUTE_CLIPPED

Clip any primitives in the buffer that are outside or partially outside the viewport.

D3DEXECUTE_UNCLIPPED

All primitives in the buffer are contained within the viewport.

See also **D3DEXECUTEDATA**, **D3DINSTRUCTION**, **IDirect3DExecuteBuffer::Validate**

IDirect3DDevice::GetCaps

```
HRESULT GetCaps(LPD3DDEVICEDESC lpD3DHWDevDesc,
    LPD3DDEVICEDESC lpD3DHELDevDesc);
```

Retrieves the capabilities of the Direct3DDevice object.

- Returns D3D_OK if successful, or an error otherwise, which may be one of the following values:

  DDERR_INVALIDOBJECT

  DDERR_INVALIDPARAMS

lpD3DHWDevDesc
Address of the **D3DDEVICEDESC** structure that will contain the hardware features of the device.

lpD3DHELDevDesc
Address of the **D3DDEVICEDESC** structure that will contain the software emulation being provided.

This method does not retrieve the capabilities of the display device. To retrieve this information, use the **IDirectDraw::GetCaps** method.

IDirect3DDevice::GetDirect3D

```
HRESULT GetDirect3D(LPDIRECT3D* lpD3D);
```

Retrieves the current **IDirect3D** interface.

- Returns D3D_OK if successful, or an error otherwise. For a list of possible return codes, see **Direct3D Immediate-Mode Return Values**.

lpD3D
Address that will contain the interface when the method returns.

IDirect3DDevice::GetMatrix

```
HRESULT GetMatrix(D3DMATRIXHANDLE lpD3DMatHandle,
    LPD3DMATRIX lpD3DMatrix);
```

Retrieves a matrix from a matrix handle. This matrix handle must have been created by using the **IDirect3DDevice::CreateMatrix** method.

- Returns D3D_OK if successful, or an error otherwise, such as **DDERR_INVALIDPARAMS**.

lpD3DMatHandle
Address of a variable that contains the matrix to be retrieved.

lpD3DMatrix
Address of a **D3DMATRIX** structure that contains the matrix when the method returns.

See also **IDirect3DDevice::CreateMatrix**, **IDirect3DDevice::DeleteMatrix**, **IDirect3DDevice::SetMatrix**

IDirect3DDevice::GetPickRecords

```
HRESULT GetPickRecords(LPDWORD lpCount,
    LPD3DPICKRECORD lpD3DPickRec);
```

Retrieves the pick records for a device.

- Returns D3D_OK if successful or an error otherwise.

lpCount
Address of a variable that contains the number of **D3DPICKRECORD** structures to retrieve.

lpD3DPickRec
Address that will contain an array of **D3DPICKRECORD** structures when the method returns.

An application typically calls this method twice. In the first call, the second parameter is set to NULL, and the first parameter retrieves a count of all relevant **D3DPICKRECORD** structures. The application then allocates sufficient memory for those structures and calls the method again, specifying the newly allocated memory for the second parameter.

IDirect3DDevice::GetStats

```
HRESULT GetStats(LPD3DSTATS lpD3DStats);
```

Retrieves statistics about a device.

- Returns D3D_OK if successful, or an error otherwise, which may be one of the following values:

  DDERR_INVALIDOBJECT

  DDERR_INVALIDPARAMS

lpD3DStats
Address of a **D3DSTATS** structure that will be filled with the statistics.

IDirect3DDevice::Initialize

```
HRESULT Initialize(LPDIRECT3D lpd3d, LPGUID lpGUID,
    LPD3DDEVICEDESC lpd3ddvdesc);
```

Initializes a device.

- Returns D3D_OK if successful, or an error otherwise. For a list of possible return values, see **Direct3D Immediate-Mode Return Values**.

lpd3d
Address of the Direct3D device to use as an initializer.

lpGUID
Address of the globally unique identifier (GUID) used as the interface identifier.

lpd3ddvdesc
Address of a **D3DDEVICEDESC** structure describing the Direct3DDevice object to be initialized.

IDirect3DDevice::NextViewport

```
HRESULT NextViewport(LPDIRECT3DVIEWPORT lpDirect3DViewport,
    LPDIRECT3DVIEWPORT* lplpDirect3DViewport, DWORD dwFlags);
```

Enumerates the viewports associated with the device.

- Returns D3D_OK if successful, or an error otherwise, which may be one of the following values:

  DDERR_INVALIDOBJECT

  DDERR_INVALIDPARAMS

lpDirect3DViewport
Address of a viewport in the list of viewports associated with this Direct3DDevice object.

lplpDirect3DViewport
Address of the next viewport in the list of viewports associated with this Direct3DDevice object.

dwFlags
Flags specifying which viewport to retrieve from the list of viewports. The default setting is D3DNEXT_NEXT.

D3DNEXT_HEAD	Retrieve the item at the beginning of the list.
D3DNEXT_NEXT	Retrieve the next item in the list.
D3DNEXT_TAIL	Retrieve the item at the end of the list.

IDirect3DDevice::Pick

```
HRESULT Pick(LPDIRECT3DEXECUTEBUFFER lpDirect3DExecuteBuffer,
    LPDIRECT3DVIEWPORT lpDirect3DViewport, DWORD dwFlags,
    LPD3DRECT lpRect);
```

Executes a buffer without performing any rendering, but returns a z-ordered list of offsets to the primitives that cover the rectangle specified by *lpRect*.

This call fails if the Direct3DExecuteBuffer object is locked.

- Returns D3D_OK if successful, or an error otherwise, which may be one of the following values:

  D3DERR_EXECUTE_LOCKED

  DDERR_INVALIDOBJECT

  DDERR_INVALIDPARAMS

lpDirect3DExecuteBuffer
Address of an execute buffer from which the z-ordered list is retrieved.

lpDirect3DViewport
Address of a viewport in the list of viewports associated with this Direct3DDevice object.

dwFlags
No flags are currently defined for this method.

lpRect
Address of a **D3DRECT** structure specifying the range of device coordinates to be picked.

If the **x1** and **x2** members of the structure specified in the *lpRect* parameter are equal, and the **y1** and **y2** members are equal, a single pixel is used for picking. The coordinates are specified in device-pixel space.

All Direct3DExecuteBuffer objects must be attached to a Direct3DDevice object in order for this method to succeed.

See also **IDirect3DDevice::GetPickRecords**

IDirect3DDevice::QueryInterface

```
HRESULT QueryInterface(REFIID riid, LPVOID* obp);
```

Determines if the Direct3DDevice object supports a particular COM interface. If it does, the system increases the reference count for the object, and the application can begin using that interface immediately. This method is part of the **IUnknown** interface inherited by Direct3DDevice.

- Returns D3D_OK if successful, or an error otherwise, which may be one of the following values:

  DDERR_INVALIDOBJECT

  DDERR_INVALIDPARAMS

riid
Reference identifier of the interface being requested.

obp
Address of a pointer that will be filled with the interface pointer if the query is successful.

If the application does not need to use the interface retrieved by a call to this method, it must call the **Release** method for that interface to free it. The **IDirect3DDevice::QueryInterface** method allows Direct3DDevice objects to be extended by Microsoft and third parties without interfering with each other's existing or future functionality.

IDirect3DDevice::Release

```
ULONG Release();
```

Decreases the reference count of the Direct3DDevice object by 1. This method is part of the **IUnknown** interface inherited by Direct3DDevice.

- Returns the new reference count of the object.

The Direct3DDevice object deallocates itself when its reference count reaches 0. Use the **IDirect3DDevice::AddRef** method to increase the reference count of the object by 1.

IDirect3DDevice::SetMatrix

```
HRESULT SetMatrix(D3DMATRIXHANDLE d3dMatHandle,
    LPD3DMATRIX lpD3DMatrix);
```

Applies a matrix to a matrix handle. This matrix handle must have been created by using the **IDirect3DDevice::CreateMatrix** method.

- Returns D3D_OK if successful, or an error otherwise, such as **DDERR_INVALIDPARAMS**.

d3dMatHandle
Matrix handle to be set.

lpD3DMatrix
Address of a **D3DMATRIX** structure that describes the matrix to be set.

Transformations inside the execute buffer include a handle of a matrix. The **IDirect3DDevice::SetMatrix** method enables an application to change this matrix without having to lock and unlock the execute buffer.

See also **IDirect3DDevice::CreateMatrix**, **IDirect3DDevice::GetMatrix**, **IDirect3DDevice::DeleteMatrix**

IDirect3DDevice::SwapTextureHandles

```
HRESULT SwapTextureHandles(LPDIRECT3DTEXTURE lpD3DTex1,
    LPDIRECT3DTEXTURE lpD3DTex2);
```

Swaps two texture handles.

- Returns D3D_OK if successful or an error otherwise.

lpD3DTex1 and *lpD3DTex2*
Addresses of the textures whose handles will be swapped when the method returns.

This method is useful when an application is changing all the textures in a complicated object.

IDirect3DExecuteBuffer Interface

IDirect3DExecuteBuffer Interface Method Groups

Your application uses the methods of the **IDirect3DExecuteBuffer** interface to set up and control a Direct3D execute buffer. The methods can be organized into the following groups:

Execute data	**GetExecuteData**
	SetExecuteData
IUnknown	**AddRef**
	QueryInterface
	Release

Lock and unlock	**Lock**
	Unlock
Miscellaneous	**Initialize**
	Optimize
	Validate

All COM interfaces inherit the **IUnknown** interface methods, which are listed in the "IUnknown" group above. These three methods allow additional interfaces to be added to the Direct3DExecuteBuffer object without affecting the functionality of the original interface.

IDirect3DExecuteBuffer::AddRef

```
ULONG AddRef();
```

Increases the reference count of the Direct3DExecuteBuffer object by 1. This method is part of the **IUnknown** interface inherited by Direct3DExecuteBuffer.

- Returns the new reference count of the object.

When the Direct3DExecuteBuffer object is created, its reference count is set to 1. Every time an application obtains an interface to the object or calls the **AddRef** method, the object's reference count is increased by 1. Use the **IDirect3DExecuteBuffer::Release** method to decrease the reference count of the object by 1.

IDirect3DExecuteBuffer::GetExecuteData

```
HRESULT GetExecuteData(LPD3DEXECUTEDATA lpData);
```

Retrieves the execute data state of the Direct3DExecuteBuffer object. The execute data is used to describe the contents of the Direct3DExecuteBuffer object.

- Returns D3D_OK if successful, or an error otherwise, which may be one of the following values:

  D3DERR_EXECUTE_LOCKED

  DDERR_INVALIDOBJECT

  DDERR_INVALIDPARAMS

lpData
Address of a **D3DEXECUTEDATA** structure that will be filled with the current execute data state of the Direct3DExecuteBuffer object.

This call fails if the Direct3DExecuteBuffer object is locked.

See also **IDirect3DExecuteBuffer::SetExecuteData**

IDirect3DExecuteBuffer::Initialize

```
HRESULT Initialize(LPDIRECT3DDEVICE lpDirect3DDevice,
    LPD3DEXECUTEBUFFERDESC lpDesc);
```

This method is provided for compliance with the COM protocol.

- Returns **DDERR_ALREADYINITIALIZED** because the Direct3DExecuteBuffer object is initialized when it is created.

lpDirect3DDevice
Address of the device representing the Direct3D object.

lpDesc
Address of a **D3DEXECUTEBUFFERDESC** structure that describes the Direct3DExecuteBuffer object to be created. The call fails if a buffer of at least the specified size cannot be created.

IDirect3DExecuteBuffer::Lock

```
HRESULT Lock(LPD3DEXECUTEBUFFERDESC lpDesc);
```

Obtains a direct pointer to the commands in the execute buffer.

- Returns D3D_OK if successful, or an error otherwise, which may be one of the following values:

  D3DERR_EXECUTE_LOCKED

  DDERR_INVALIDOBJECT

  DDERR_INVALIDPARAMS

  DDERR_WASSTILLDRAWING

lpDesc
Address of a **D3DEXECUTEBUFFERDESC** structure. When the method returns, the **lpData** member will be set to point to the actual data the application has access to. This data may reside in system or video memory, and is specified by the **dwCaps** member. The application may use the **IDirect3DExecuteBuffer::Lock** method to request that Direct3D move the data between system or video memory.

This call fails if the Direct3DExecuteBuffer object is locked—that is, if another thread is accessing the buffer, or if a **IDirect3DDevice::Execute** method that was issued on this buffer has not yet completed.

See also **IDirect3DExecuteBuffer::Unlock**

IDirect3DExecuteBuffer::Optimize

```
HRESULT Optimize();
```

Not currently supported.

IDirect3DExecuteBuffer::QueryInterface

```
HRESULT QueryInterface(REFIID riid, LPVOID* ppvObj);
```

Determines if the Direct3DExecuteBuffer object supports a particular COM interface. If it does, the system increases the reference count for the object, and the application can begin using that interface immediately. This method is part of the **IUnknown** interface inherited by Direct3DExecuteBuffer.

- Returns D3D_OK if successful, or an error otherwise, which may be one of the following values:

  DDERR_INVALIDOBJECT

  DDERR_INVALIDPARAMS

riid
Reference identifier of the interface being requested.

ppvObj
Address of a pointer that will be filled with the interface pointer if the query is successful.

If the application does not need to use the interface retrieved by a call to this method, it must call the **Release** method for that interface to free it. The **IDirect3DExecuteBuffer::QueryInterface** method allows Direct3DExecuteBuffer objects to be extended by Microsoft and third parties without interfering with each other's existing or future functionality.

This call fails if the Direct3DExecuteBuffer object is locked.

IDirect3DExecuteBuffer::Release

```
ULONG Release();
```

Decreases the reference count of the Direct3DExecuteBuffer object by 1. This method is part of the **IUnknown** interface inherited by Direct3DExecuteBuffer.

- Returns the new reference count of the object.

The Direct3DExecuteBuffer object deallocates itself when its reference count reaches 0. Use the **IDirect3DExecuteBuffer::AddRef** method to increase the reference count of the object by 1.

This call fails if the Direct3DExecuteBuffer object is locked.

IDirect3DExecuteBuffer::SetExecuteData

```
HRESULT SetExecuteData(LPD3DEXECUTEDATA lpData);
```

Sets the execute data state of the Direct3DExecuteBuffer object. The execute data is used to describe the contents of the Direct3DExecuteBuffer object.

- Returns D3D_OK if successful, or an error otherwise, which may be one of the following values:

  D3DERR_EXECUTE_LOCKED

  DDERR_INVALIDOBJECT

  DDERR_INVALIDPARAMS

lpData
Address of a **D3DEXECUTEDATA** structure that describes the execute buffer layout.

This call fails if the Direct3DExecuteBuffer object is locked.

See also **IDirect3DExecuteBuffer::GetExecuteData**

IDirect3DExecuteBuffer::Unlock

```
HRESULT Unlock();
```

Releases the direct pointer to the commands in the execute buffer. This must be done prior to calling the **IDirect3DDevice::Execute** method for the buffer.

- Returns D3D_OK if successful, or an error otherwise, which may be one of the following values:

  D3DERR_EXECUTE_NOT_LOCKED

  DDERR_INVALIDOBJECT

See also **IDirect3DExecuteBuffer::Lock**

IDirect3DExecuteBuffer::Validate

```
HRESULT Validate(LPDWORD lpdwOffset, LPD3DVALIDATECALLBACK lpFunc,
    LPVOID lpUserArg, DWORD dwReserved);
```

Checks an execute buffer and returns an offset into the buffer when any errors are encountered. This method is a debugging routine.

- Returns D3D_OK if successful, or an error otherwise, which may be one of the following values:

  DDERR_INVALIDOBJECT

  DDERR_INVALIDPARAMS

lpdwOffset
Address of a variable that will be filled with the offset into the execute buffer at which an error was first detected. This parameter is filled in only if NULL is specified for the *lpFunc* parameter. If a callback function is specified for *lpFunc*, the offset for each error is passed to the callback function, and the *lpdwOffset* parameter is not set.

lpFunc
Address of an application-defined **D3DVALIDATECALLBACK** callback function. If this parameter is NULL, checking stops when the first error is detected.

lpUserArg
Address of application-defined data passed to the callback function.

dwReserved
Reserved for future use.

The callback function specified in the *lpFunc* parameter is called whenever an error is detected in the execute buffer. The system passes to this callback function the value specified in *lpUserArg* and the offset into the execute buffer where the error was detected.

This call fails if the Direct3DExecuteBuffer object is locked.

IDirect3DLight Interface

IDirect3DLight Interface Method Groups

Applications use the methods of the **IDirect3DLight** interface to retrieve and set the capabilities of lights. The methods can be organized into the following groups:

Get and set	**GetLight**
	SetLight
Initialization	**Initialize**
IUnknown	**AddRef**
	QueryInterface
	Release

All COM interfaces inherit the **IUnknown** interface methods, which are listed in the "IUnknown" group above. These three methods allow additional interfaces to be added to the Direct3DLight object without affecting the functionality of the original interface.

IDirect3DLight::AddRef

```
ULONG AddRef();
```

Increases the reference count of the Direct3DLight object by 1. This method is part of the **IUnknown** interface inherited by Direct3DLight.

- Returns the new reference count of the object.

When the Direct3DLight object is created, its reference count is set to 1. Every time an application obtains an interface to the object or calls the **AddRef** method, the object's reference count is increased by 1. Use the **IDirect3DLight::Release** method to decrease the reference count of the object by 1.

IDirect3DLight::GetLight

```
HRESULT GetLight(LPD3DLIGHT lpLight);
```

Retrieves the light information for the Direct3DLight object.

- Returns D3D_OK if successful, or an error otherwise, which may be one of the following values:

  DDERR_INVALIDOBJECT

  DDERR_INVALIDPARAMS

lpLight
Address of a **D3DLIGHT** structure that will be filled with the current light data.

See also **IDirect3DLight::SetLight**

IDirect3DLight::Initialize

```
HRESULT Initialize(LPDIRECT3D lpDirect3D);
```

This method is provided for compliance with the COM protocol.

- Returns **DDERR_ALREADYINITIALIZED** because the Direct3DLight object is initialized when it is created.

lpDirect3D
Address of the Direct3D structure representing the Direct3D object.

IDirect3DLight::QueryInterface

```
HRESULT QueryInterface(REFIID riid, LPVOID* ppvObj);
```

Determines if the Direct3DLight object supports a particular COM interface. If it does, the system increases the reference count for the object, and the application can begin using that interface immediately. This method is part of the **IUnknown** interface inherited by Direct3DLight.

- Returns D3D_OK if successful, or an error otherwise, which may be one of the following values:

  DDERR_INVALIDOBJECT

  DDERR_INVALIDPARAMS

riid
Reference identifier of the interface being requested.

ppvObj
Address of a pointer that will be filled with the interface pointer if the query is successful.

If the application does not need to use the interface retrieved by a call to this method, it must call the **Release** method for that interface to free it. The **IDirect3DLight::QueryInterface** method allows Direct3DLight objects to be extended by Microsoft and third parties without interfering with each other's existing or future functionality.

IDirect3DLight::Release

```
ULONG Release();
```

Decreases the reference count of the Direct3DLight object by 1. This method is part of the **IUnknown** interface inherited by Direct3DLight.

- Returns the new reference count of the object.

The Direct3DLight object deallocates itself when its reference count reaches 0. Use the **IDirect3DLight::AddRef** method to increase the reference count of the object by 1.

IDirect3DLight::SetLight

```
HRESULT SetLight(LPD3DLIGHT lpLight);
```

Sets the light information for the Direct3DLight object.

- Returns D3D_OK if successful, or an error otherwise, which may be one of the following values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS

lpLight
Address of a **D3DLIGHT** structure that will be used to set the current light data.

See also **IDirect3DLight::GetLight**

IDirect3DMaterial Interface

IDirect3DMaterial Interface Method Groups

Applications use the methods of the **IDirect3DMaterial** interface to retrieve and set the properties of materials. The methods can be organized into the following groups:

Color reservation	**Reserve**
	Unreserve
IUnknown	**AddRef**
	QueryInterface
	Release
Materials	**GetMaterial**
	SetMaterial
Miscellaneous	**GetHandle**
	Initialize

All COM interfaces inherit the **IUnknown** interface methods, which are listed in the "IUnknown" group above. These three methods allow additional interfaces to be added to the Direct3DMaterial object without affecting the functionality of the original interface.

IDirect3DMaterial::AddRef

```
ULONG AddRef();
```

Increases the reference count of the Direct3DMaterial object by 1. This method is part of the **IUnknown** interface inherited by Direct3DMaterial.

- Returns the new reference count of the object.

When the Direct3DMaterial object is created, its reference count is set to 1. Every time an application obtains an interface to the object or calls the **AddRef** method, the object's reference count is increased by 1. Use the **IDirect3DMaterial::Release** method to decrease the reference count of the object by 1.

IDirect3DMaterial::GetHandle

```
HRESULT GetHandle(LPDIRECT3DDEVICE lpDirect3DDevice,
    LPD3DMATERIALHANDLE lpHandle);
```

Obtains the material handle for the Direct3DMaterial object. This handle is used in all Direct3D API calls where a material is to be referenced. A material can be used by only one device at a time.

If the device is destroyed, the material is disassociated from the device.

- Returns D3D_OK if successful, or **DDERR_INVALIDOBJECT** otherwise.

lpDirect3DDevice
Address of the Direct3DDevice object in which the material is being used.

lpHandle
Address of a variable that will be filled with the material handle corresponding to the Direct3DMaterial object.

IDirect3DMaterial::GetMaterial

```
HRESULT GetMaterial(LPD3DMATERIAL lpMat);
```

Retrieves the material data for the Direct3DMaterial object.

- Returns D3D_OK if successful, or an error otherwise, which may be one of the following values:

  DDERR_INVALIDOBJECT

  DDERR_INVALIDPARAMS

lpMat
Address of a **D3DMATERIAL** structure that will be filled with the current material properties.

See also **IDirect3DMaterial::SetMaterial**

IDirect3DMaterial::Initialize

```
HRESULT Initialize(LPDIRECT3D lpDirect3D);
```

This method is provided for compliance with the COM protocol.

- Returns **DDERR_ALREADYINITIALIZED** because the Direct3DMaterial object is initialized when it is created.

lpDirect3D
Address of the Direct3D structure representing the Direct3D object.

IDirect3DMaterial::QueryInterface

```
HRESULT QueryInterface(REFIID riid, LPVOID* ppvObj);
```

Determines if the Direct3DMaterial object supports a particular COM interface. If it does, the system increases the reference count for the object, and the application can begin using that interface immediately. This method is part of the **IUnknown** interface inherited by Direct3DMaterial.

- Returns D3D_OK if successful, or an error otherwise, which may be one of the following values:

  DDERR_INVALIDOBJECT

  DDERR_INVALIDPARAMS

riid
Reference identifier of the interface being requested.

ppvObj
Address of a pointer that will be filled with the interface pointer if the query is successful.

If the application does not need to use the interface retrieved by a call to this method, it must call the **Release** method for that interface to free it. The **IDirect3DMaterial::QueryInterface** method allows Direct3DMaterial objects to be extended by Microsoft and third parties without interfering with each other's existing or future functionality.

IDirect3DMaterial::Release

```
ULONG Release();
```

Decreases the reference count of the Direct3DMaterial object by 1. This method is part of the **IUnknown** interface inherited by Direct3DMaterial.

- Returns the new reference count of the object.

The Direct3DMaterial object deallocates itself when its reference count reaches 0. Use the **IDirect3DMaterial::AddRef** method to increase the reference count of the object by 1.

IDirect3DMaterial::Reserve

```
HRESULT Reserve();
```

Not currently implemented.

IDirect3DMaterial::SetMaterial

```
HRESULT SetMaterial(LPD3DMATERIAL lpMat);
```

Sets the material data for the Direct3DMaterial object.

- Returns D3D_OK if successful, or an error otherwise, which may be one of the following values:

  DDERR_INVALIDOBJECT

  DDERR_INVALIDPARAMS

lpMat
Address of a **D3DMATERIAL** structure that contains the material properties.

See also **IDirect3DMaterial::GetMaterial**

IDirect3DMaterial::Unreserve

```
HRESULT Unreserve();
```

Not currently implemented.

IDirect3DTexture Interface

IDirect3DTexture Interface Method Groups

Applications use the methods of the **IDirect3DTexture** interface to retrieve and set the properties of textures. The methods can be organized into the following groups:

Handles	**GetHandle**
Initialization	**Initialize**
IUnknown	**AddRef** **QueryInterface** **Release**
Loading	**Load**

	Unload
Palette information	**PaletteChanged**

All COM interfaces inherit the **IUnknown** interface methods, which are listed in the "IUnknown" group above. These three methods allow additional interfaces to be added to the Direct3DTexture object without affecting the functionality of the original interface.

The Direct3DTexture object is obtained through the appropriate call to the **IDirect3D::QueryInterface** method from a DirectDrawSurface object that was created as a texture map.

IDirect3DTexture::AddRef

```
ULONG AddRef();
```

Increases the reference count of the Direct3DTexture object by 1. This method is part of the **IUnknown** interface inherited by Direct3DTexture.

- Returns the new reference count of the object.

When the Direct3DTexture object is created, its reference count is set to 1. Every time an application obtains an interface to the object or calls the **AddRef** method, the object's reference count is increased by 1. Use the **IDirect3DTexture::Release** method to decrease the reference count of the object by 1.

IDirect3DTexture::GetHandle

```
HRESULT GetHandle(LPDIRECT3DDEVICE lpDirect3DDevice,
    LPD3DTEXTUREHANDLE lpHandle);
```

Obtains the texture handle for the Direct3DTexture object. This handle is used in all Direct3D API calls where a texture is to be referenced.

- Returns D3D_OK if successful, or an error otherwise, which may be one of the following values:

  D3DERR_BADOBJECT

  DDERR_INVALIDPARAMS

lpDirect3DDevice
Address of the Direct3DDevice object into which the texture is to be loaded.

lpHandle
Address that will contain the texture handle corresponding to the Direct3DTexture object.

IDirect3DTexture::Initialize

```
HRESULT Initialize(LPDIRECT3DDEVICE lpD3DDevice,
    LPDIRECTDRAWSURFACE lpDDSurface);
```

This method is provided for compliance with the COM protocol.

- Returns **DDERR_ALREADYINITIALIZED** because the Direct3DTexture object is initialized when it is created.

lpD3DDevice
Address of the device representing the Direct3D object.

lpDDSurface
Address of the DirectDraw surface for this object.

IDirect3DTexture::Load

```
HRESULT Load(LPDIRECT3DTEXTURE lpD3DTexture);
```

Loads a texture that was created with the DDSCAPS_ALLOCONLOAD flag, which indicates that memory for the DirectDraw surface is not allocated until the surface is loaded by using this method.

- Returns D3D_OK if successful, or an error otherwise. For a list of possible return values, see **Direct3D Immediate-Mode Return Values**.

lpD3DTexture
Address of the texture to load.

See also **IDirect3DTexture::Unload**

IDirect3DTexture::PaletteChanged

```
HRESULT PaletteChanged(DWORD dwStart, DWORD dwCount);
```

Informs the driver that the palette has changed on a surface.

- Returns D3D_OK if successful, or an error otherwise. For a list of possible return values, see **Direct3D Immediate-Mode Return Values**.

dwStart
Index of first palette entry that has changed.

dwCount
Number of palette entries that have changed.

This method is particularly useful for applications that play video clips and therefore require palette-changing capabilities.

IDirect3DTexture::QueryInterface

```
HRESULT QueryInterface(REFIID riid, LPVOID* ppvObj);
```

Determines if the Direct3DTexture object supports a particular COM interface. If it does, the system increases the reference count for the object, and the application can begin using that interface immediately. This method is part of the **IUnknown** interface inherited by Direct3DTexture.

- Returns D3D_OK if successful, or an error otherwise, which may be one of the following values:

  DDERR_INVALIDOBJECT

  DDERR_INVALIDPARAMS

riid
Reference identifier of the interface being requested.

ppvObj
Address of a pointer that will be filled with the interface pointer if the query is successful.

If the application does not need to use the interface retrieved by a call to this method, it must call the **Release** method for that interface to free it. The **IDirect3DTexture::QueryInterface** method allows Direct3DTexture objects to be extended by Microsoft and third parties without interfering with each other's existing or future functionality.

IDirect3DTexture::Release

```
ULONG Release();
```

Decreases the reference count of the Direct3DTexture object by 1. This method is part of the **IUnknown** interface inherited by Direct3DTexture.

- Returns the new reference count of the object.

The Direct3DTexture object deallocates itself when its reference count reaches 0. Use the **IDirect3DTexture::AddRef** method to increase the reference count of the object by 1.

IDirect3DTexture::Unload

```
HRESULT Unload();
```

Unloads the current texture.

- Returns D3D_OK if successful, or an error otherwise. For a list of possible return values, see **Direct3D Immediate-Mode Return Values**.

See also **IDirect3DTexture::Load**

IDirect3DViewport Interface

IDirect3DViewport Interface Method Groups

Applications use the methods of the **IDirect3DViewport** interface to retrieve and set the properties of viewports. The methods can be organized into the following groups:

Backgrounds	**GetBackground**
	GetBackgroundDepth
	SetBackground
	SetBackgroundDepth
Initialization	**Initialize**
IUnknown	**AddRef**
	QueryInterface
	Release
Lights	**AddLight**
	DeleteLight
	LightElements
	NextLight
Materials and viewports	**Clear**
	GetViewport
	SetViewport
Transformation	**TransformVertices**

All COM interfaces inherit the **IUnknown** interface methods, which are listed in the "IUnknown" group above. These three methods allow additional interfaces to

be added to the Direct3DViewport object without affecting the functionality of the original interface.

IDirect3DViewport::AddLight

```
HRESULT AddLight(LPDIRECT3DLIGHT lpDirect3DLight);
```

Adds the specified light to the list of Direct3DLight objects associated with this viewport.

- Returns D3D_OK if successful, or an error otherwise, which may be one of the following values:

  DDERR_INVALIDOBJECT

  DDERR_INVALIDPARAMS

lpDirect3DLight
Address of the Direct3DLight object that should be associated with this Direct3DDevice object.

IDirect3DViewport::AddRef

```
ULONG AddRef();
```

Increases the reference count of the Direct3DViewport object by 1. This method is part of the **IUnknown** interface inherited by Direct3DViewport.

- Returns the new reference count of the object.

When the Direct3DViewport object is created, its reference count is set to 1. Every time an application obtains an interface to the object or calls the **AddRef** method, the object's reference count is increased by 1. Use the **IDirect3DViewport::Release** method to decrease the reference count of the object by 1.

IDirect3DViewport::Clear

```
HRESULT Clear(DWORD dwCount, LPD3DRECT lpRects, DWORD dwFlags);
```

Clears the viewport or a set of rectangles in the viewport to the current background material.

- Returns D3D_OK if successful, or an error otherwise, which may be one of the following values:

  DDERR_INVALIDOBJECT

  DDERR_INVALIDPARAMS

dwCount
Number of rectangles pointed to by *lpRects*.

lpRects
Address of an array of **D3DRECT** structures.

dwFlags
Flags indicating what to clear: the rendering target, the z-buffer, or both.

D3DCLEAR_TARGET	Clear the rendering target to the background material (if set).
D3DCLEAR_ZBUFFER	Clear the z-buffer or set it to the current background depth field (if set).

IDirect3DViewport::DeleteLight

```
HRESULT DeleteLight(LPDIRECT3DLIGHT lpDirect3DLight);
```

Removes the specified light from the list of Direct3DLight objects associated with this viewport.

- Returns D3D_OK if successful, or an error otherwise, which may be one of the following values:

  DDERR_INVALIDOBJECT

  DDERR_INVALIDPARAMS

lpDirect3DLight
Address of the Direct3DLight object that should be disassociated with this Direct3DDevice object.

IDirect3DViewport::GetBackground

```
HRESULT GetBackground(LPD3DMATERIALHANDLE lphMat, LPBOOL lpValid);
```

Retrieves the handle of a material that represents the current background associated with the viewport.

- Returns D3D_OK if successful, or an error otherwise, which may be one of the following values:

  DDERR_INVALIDOBJECT

  DDERR_INVALIDPARAMS

lphMat
Address that will contain the handle of the material being used as the background.

lpValid
Address of a variable that will be filled to indicate whether a background is associated with the viewport. If this parameter is FALSE, no background is associated with the viewport.

See also **IDirect3DViewport::SetBackground**

IDirect3DViewport::GetBackgroundDepth

```
HRESULT GetBackgroundDepth(LPDIRECTDRAWSURFACE* lplpDDSurface,
    LPBOOL lpValid);
```

Retrieves a DirectDraw surface that represents the current background-depth field associated with the viewport.

- Returns D3D_OK if successful, or an error otherwise, which may be one of the following values:

  DDERR_INVALIDOBJECT

  DDERR_INVALIDPARAMS

lplpDDSurface
Address that will be initialized to point to a DirectDrawSurface object representing the background depth.

lpValid
Address of a variable that is set to FALSE if no background depth is associated with the viewport.

See also **IDirect3DViewport::SetBackgroundDepth**

IDirect3DViewport::GetViewport

```
HRESULT GetViewport(LPD3DVIEWPORT lpData);
```

Retrieves the viewport registers of the viewport.

- Returns D3D_OK if successful, or an error otherwise, which may be one of the following values:

  DDERR_INVALIDOBJECT

  DDERR_INVALIDPARAMS

lpData
Address of a **D3DVIEWPORT** structure representing the viewport.

See also **IDirect3DViewport::SetViewport**

IDirect3DViewport::Initialize

```
HRESULT Initialize(LPDIRECT3D lpDirect3D);
```

This method is provided for compliance with the COM protocol.

- Returns **DDERR_ALREADYINITIALIZED** because the Direct3DViewport object is initialized when it is created.

lpDirect3D
Address of the Direct3D structure representing the Direct3D object.

IDirect3DViewport::LightElements

```
HRESULT LightElements(DWORD dwElementCount, LPD3DLIGHTDATA lpData);
```

Calculates light intensities and colors for rendering a geometry.

- Returns D3D_OK if successful, or an error otherwise, which may be one of the following values:

  DDERR_INVALIDOBJECT

  DDERR_INVALIDPARAMS

dwElementCount
Number of elements to be lit.

lpData
Address of a **D3DLIGHTDATA** structure that contains the points to be lit and the resulting colors.

IDirect3DViewport::NextLight

```
HRESULT NextLight(LPDIRECT3DLIGHT lpDirect3DLight,
    LPDIRECT3DLIGHT* lplpDirect3DLight, DWORD dwFlags);
```

Enumerates the Direct3DLight objects associated with the viewport.

- Returns D3D_OK if successful, or an error otherwise, which may be one of the following values:

  DDERR_INVALIDOBJECT

  DDERR_INVALIDPARAMS

lpDirect3DLight
Address of a light in the list of lights associated with this Direct3DDevice object.

lplpDirect3DLight
Address of a pointer that will contain the requested light in the list of lights associated with this Direct3DDevice object. The requested light is specified in the *dwFlags* parameter.

dwFlags
Flags specifying which light to retrieve from the list of lights. The default setting is D3DNEXT_NEXT.

D3DNEXT_HEAD	Retrieve the item at the beginning of the list.
D3DNEXT_NEXT	Retrieve the next item in the list.
D3DNEXT_TAIL	Retrieve the item at the end of the list.

IDirect3DViewport::QueryInterface

```
HRESULT QueryInterface(REFIID riid, LPVOID* ppvObj);
```

Determines if the Direct3DViewport object supports a particular COM interface. If it does, the system increases the reference count for the object, and the application can begin using that interface immediately. This method is part of the **IUnknown** interface inherited by Direct3DViewport.

- Returns D3D_OK if successful, or an error otherwise, which may be one of the following values:

  DDERR_INVALIDOBJECT

  DDERR_INVALIDPARAMS

riid
Reference identifier of the interface being requested.

ppvObj
Address of a pointer that will be filled with the interface pointer if the query is successful.

If the application does not need to use the interface retrieved by a call to this method, it must call the **Release** method for that interface to free it. The **IDirect3DViewport::QueryInterface** method allows Direct3DViewport objects to be extended by Microsoft and third parties without interfering with each other's existing or future functionality.

IDirect3DViewport::Release

```
ULONG Release();
```

Decreases the reference count of the Direct3DViewport object by 1. This method is part of the **IUnknown** interface inherited by Direct3DViewport.

- Returns the new reference count of the object.

The Direct3DViewport object deallocates itself when its reference count reaches 0. Use the **IDirect3DViewport::AddRef** method to increase the reference count of the object by 1.

IDirect3DViewport::SetBackground

```
HRESULT SetBackground(D3DMATERIALHANDLE hMat);
```

Sets the background associated with the viewport.

- Returns D3D_OK if successful, or an error otherwise, which may be one of the following values:

  DDERR_INVALIDOBJECT

  DDERR_INVALIDPARAMS

hMat
Material handle that will be used as the background.

See also **IDirect3DViewport::GetBackground**

IDirect3DViewport::SetBackgroundDepth

```
HRESULT SetBackgroundDepth(LPDIRECTDRAWSURFACE lpDDSurface);
```

Sets the background-depth field for the viewport.

- Returns D3D_OK if successful, or an error otherwise, which may be one of the following values:

  DDERR_INVALIDOBJECT

  DDERR_INVALIDPARAMS

lpDDSurface
Address of the DirectDrawSurface object representing the background depth.

The z-buffer is filled with the specified depth field when the **IDirect3DViewport::Clear** method is called and the D3DCLEAR_ZBUFFER flag is specified. The bit depth must be 16 bits.

See also **IDirect3DViewport::GetBackgroundDepth**

IDirect3DViewport::SetViewport

```
HRESULT SetViewport(LPD3DVIEWPORT lpData);
```

Sets the viewport registers of the viewport.

- Returns D3D_OK if successful, or an error otherwise, which may be one of the following values:

  DDERR_INVALIDOBJECT

  DDERR_INVALIDPARAMS

lpData
Address of a **D3DVIEWPORT** structure that contains the new viewport.

See also **IDirect3DViewport::GetViewport**

IDirect3DViewport::TransformVertices

```
HRESULT TransformVertices(DWORD dwVertexCount,
    LPD3DTRANSFORMDATA lpData, DWORD dwFlags, LPDWORD lpOffscreen);
```

Transforms a set of vertices by the transformation matrix.

- Returns D3D_OK if successful or an error otherwise, which may be one of the following values:

  DDERR_INVALIDOBJECT

  DDERR_INVALIDPARAMS

dwVertexCount
Number of vertices in the *lpData* parameter to be transformed.

lpData
Address of a **D3DTRANSFORMDATA** structure that contains the vertices to be transformed.

dwFlags
One of the following flags. See the comments section following the parameter description for a discussion of how to use these flags.

D3DTRANSFORM_CLIPPED

D3DTRANSFORM_UNCLIPPED

lpOffscreen
Address of a variable that is set to a nonzero value if the resulting vertices are all off-screen.

If the *dwFlags* parameter is set to D3DTRANSFORM_CLIPPED, this method uses the current transformation matrix to transform a set of vertices, checking the resulting vertices to see if they are within the viewing frustum. The homogeneous part of the **D3DTLVERTEX** structure within *lpData* will be set if the vertex is clipped; otherwise only the screen coordinates will be set. The clip intersection of all the vertices transformed is returned in *lpOffscreen*. That is, if *lpOffscreen* is nonzero, all the vertices were off-screen and not straddling the viewport. The

drExtent member of the **D3DTRANSFORMDATA** structure will also be set to the 2D bounding rectangle of the resulting vertices.

If the *dwFlags* parameter is set to D3DTRANSFORM_UNCLIPPED, this method uses the current transformation matrix to transform a set of vertices. In this case, the system assumes that all the resulting coordinates will be within the viewing frustum. The **drExtent** member of the **D3DTRANSFORMDATA** structure will be set to the bounding rectangle of the resulting vertices.

The **dwClip** member of **D3DTRANSFORMDATA** can help the transformation module determine whether the geometry will need clipping against the viewing volume. Before transforming a geometry, high-level software often can test whether bounding boxes or bounding spheres are wholly within the viewing volume, allowing clipping tests to be skipped, or wholly outside the viewing volume, allowing the geometry to be skipped entirely.

Structures

D3DBRANCH

```
typedef struct _D3DBRANCH {
    DWORD dwMask;
    DWORD dwValue;
    BOOL  bNegate;
    DWORD dwOffset;
} D3DBRANCH, *LPD3DBRANCH;
```

Performs conditional operations inside an execute buffer. This structure is a forward-branch structure.

dwMask
Bitmask for the branch. This mask is combined with the driver-status mask by using the logical **AND** operator. If the result equals the value specified in the **dwValue** member and the **bNegate** member is FALSE, the branch is taken.

For a list of the available driver-status masks, see the **dwStatus** member of the **D3DSTATUS** structure.

dwValue
Application-defined value to compare against the operation described in the **dwMask** member.

bNegate
TRUE to negate comparison.

dwOffset
How far to branch forward. Specify zero to exit.

D3DCOLORVALUE

```
typedef struct _D3DCOLORVALUE {
    union {
        D3DVALUE r;
        D3DVALUE dvR;
    };
    union {
        D3DVALUE g;
        D3DVALUE dvG;
    };
    union {
        D3DVALUE b;
        D3DVALUE dvB;
    };
    union {
        D3DVALUE a;
        D3DVALUE dvA;
    };
} D3DCOLORVALUE;
```

Describes color values for the **D3DLIGHT** and **D3DMATERIAL** structures.

dvR, **dvG**, **dvB**, and **dvA**
Values of the **D3DVALUE** type specifying the red, green, blue, and alpha components of a color.

D3DDEVICEDESC

```
typedef struct _D3DDeviceDesc {
    DWORD            dwSize;
    DWORD            dwFlags;
    D3DCOLORMODEL    dcmColorModel;
    DWORD            dwDevCaps;
    D3DTRANSFORMCAPS dtcTransformCaps;
    BOOL             bClipping;
    D3DLIGHTINGCAPS  dlcLightingCaps;
    D3DPRIMCAPS      dpcLineCaps;
    D3DPRIMCAPS      dpcTriCaps;
    DWORD            dwDeviceRenderBitDepth;
    DWORD            dwDeviceZBufferBitDepth;
    DWORD            dwMaxBufferSize;
    DWORD            dwMaxVertexCount;
} D3DDEVICEDESC, *LPD3DDEVICEDESC;
```

Contains a description of the current device. This structure is used to query the current device by such methods as **IDirect3DDevice::GetCaps**.

dwSize
Size, in bytes, of this structure.

dwFlags
Flags identifying the members of this structure that contain valid data.

D3DDD_BCLIPPING
The **bClipping** member is valid.

D3DDD_COLORMODEL
The **dcmColorModel** member is valid.

D3DDD_DEVCAPS
The **dwDevCaps** member is valid.

D3DDD_LIGHTINGCAPS
The **dlcLightingCaps** member is valid.

D3DDD_LINECAPS
The **dpcLineCaps** member is valid.

D3DDD_MAXBUFFERSIZE
The **dwMaxBufferSize** member is valid.

D3DDD_MAXVERTEXCOUNT
The **dwMaxVertexCount** member is valid.

D3DDD_TRANSFORMCAPS
The **dtcTransformCaps** member is valid.

D3DDD_TRICAPS
The **dpcTriCaps** member is valid.

dcmColorModel
One of the members of the **D3DCOLORMODEL** enumerated type, specifying the color model for the device.

dwDevCaps
Flags identifying the capabilities of the device.

D3DDEVCAPS_EXECUTESYSTEMMEMORY
Device can use execute buffers from system memory.

D3DDEVCAPS_EXECUTEVIDEOMEMORY
Device can use execute buffers from video memory.

D3DDEVCAPS_FLOATTLVERTEX
Device accepts floating point for post-transform vertex data.

D3DDEVCAPS_SORTDECREASINGZ
Device needs data sorted for decreasing depth.

D3DDEVCAPS_SORTEXACT
Device needs data sorted exactly.

D3DDEVCAPS_SORTINCREASINGZ

Device needs data sorted for increasing depth.

D3DDEVCAPS_TEXTURESYSTEMMEMORY

Device can retrieve textures from system memory.

D3DDEVCAPS_TEXTUREVIDEOMEMORY

Device can retrieves textures from device memory.

D3DDEVCAPS_TLVERTEXSYSTEMMEMORY

Device can use buffers from system memory for transformed and lit vertices.

D3DDEVCAPS_TLVERTEXVIDEOMEMORY

Device can use buffers from video memory for transformed and lit vertices.

dtcTransformCaps
One of the members of the **D3DTRANSFORMCAPS** structure, specifying the transformation capabilities of the device.

bClipping
TRUE if the device can perform 3D clipping.

dlcLightingCaps
One of the members of the **D3DLIGHTINGCAPS** structure, specifying the lighting capabilities of the device.

dpcLineCaps and **dpcTriCaps**
D3DPRIMCAPS structures defining the device's support for line-drawing and triangle primitives.

dwDeviceRenderBitDepth
Device's rendering bit-depth. This can be one of the following DirectDraw bit-depth constants: DDBD_8, DDBD_16, DDBD_24, or DDBD_32.

dwDeviceZBufferBitDepth
Device's z-buffer bit-depth. This can be one of the following DirectDraw bit-depth constants: DDBD_8, DDBD_16, DDBD_24, or DDBD_32.

dwMaxBufferSize
Maximum size of the execute buffer for this device. If this member is 0, the application can use any size.

dwMaxVertexCount
Maximum vertex count for this device.

See also **D3DCOLORMODEL**, **D3DFINDDEVICERESULT**, **D3DLIGHTINGCAPS**, **D3DPRIMCAPS**, **D3DTRANSFORMCAPS**

D3DEXECUTEBUFFERDESC

```
typedef struct _D3DExecuteBufferDesc {
    DWORD  dwSize;
```

```
    DWORD  dwFlags;
    DWORD  dwCaps;
    DWORD  dwBufferSize;
    LPVOID lpData;
} D3DEXECUTEBUFFERDESC;
typedef D3DEXECUTEBUFFERDESC *LPD3DEXECUTEBUFFERDESC;
```

Describes the execute buffer for such methods as **IDirect3DDevice::CreateExecuteBuffer** and **IDirect3DExecuteBuffer::Lock**.

dwSize
Size of this structure, in bytes.

dwFlags
Flags identifying the members of this structure that contain valid data.

D3DDEB_BUFSIZE	The **dwBufferSize** member is valid.
D3DDEB_CAPS	The **dwCaps** member is valid.
D3DDEB_LPDATA	The **lpData** member is valid.

dwCaps
Location in memory of the execute buffer.

D3DDEBCAPS_SYSTEMMEMORY
The execute buffer data resides in system memory.

D3DDEBCAPS_VIDEOMEMORY
The execute buffer data resides in device memory.

D3DDEBCAPS_MEM
A logical **OR** of D3DDEBCAPS_SYSTEMMEMORY and D3DDEBCAPS_VIDEOMEMORY.

dwBufferSize
Size of the execute buffer, in bytes.

lpData
Address of the buffer data.

D3DEXECUTEDATA

```
typedef struct _D3DEXECUTEDATA {
    DWORD     dwSize;
    DWORD     dwVertexOffset;
    DWORD     dwVertexCount;
    DWORD     dwInstructionOffset;
    DWORD     dwInstructionLength;
    DWORD     dwHVertexOffset;
    D3DSTATUS dsStatus;
} D3DEXECUTEDATA, *LPD3DEXECUTEDATA;
```

Specifies data for the **IDirect3DDevice::Execute** method. When this method is called and the transformation has been done, the instruction list starting at the value specified in the **dwInstructionOffset** member is parsed and rendered.

dwSize
Size of this structure, in bytes.

dwVertexOffset
Offset into the list of vertices.

dwVertexCount
Number of vertices to execute.

dwInstructionOffset
Offset into the list of instructions to execute.

dwInstructionLength
Length of the instructions to execute.

dwHVertexOffset
Offset into the list of vertices for the homogeneous vertex used when the application is supplying screen coordinate data that needs clipping.

dsStatus
Value storing the screen extent of the rendered geometry for use after the transformation is complete. This value is a **D3DSTATUS** structure.

See also **D3DSTATUS**

D3DFINDDEVICERESULT

```
typedef struct _D3DFINDDEVICERESULT {
    DWORD         dwSize;
    GUID          guid;
    D3DDEVICEDESC ddHwDesc;
    D3DDEVICEDESC ddSwDesc;
} D3DFINDDEVICERESULT, *LPD3DFINDDEVICERESULT;
```

Identifies a device an application has found by calling the **IDirect3D::FindDevice** method.

dwSize
Size, in bytes, of the structure.

guid
Globally unique identifier (GUID) of the device that was found.

ddHwDesc and **ddSwDesc**
D3DDEVICEDESC structures describing the hardware and software devices that were found.

See also **D3DFINDDEVICESEARCH**

D3DFINDDEVICESEARCH

```
typedef struct _D3DFINDDEVICESEARCH {
    DWORD         dwSize;
    DWORD         dwFlags;
    BOOL          bHardware;
    D3DCOLORMODEL dcmColorModel;
    GUID          guid;
    DWORD         dwCaps;
    D3DPRIMCAPS   dpcPrimCaps;
} D3DFINDDEVICESEARCH, *LPD3DFINDDEVICESEARCH;
```

Specifies the characteristics of a device an application wants to find. This structure is used in calls to the **IDirect3D::FindDevice** method.

dwSize
Size, in bytes, of this structure.

dwFlags
Flags defining the type of device the application wants to find. This member can be one or more of the following values:

D3DFDS_ALPHACMPCAPS

Match the **dwAlphaCmpCaps** member of the **D3DPRIMCAPS** structure specified as the **dpcPrimCaps** member of this structure.

D3DFDS_COLORMODEL

Match the color model specified in the **dcmColorModel** member of this structure.

D3DFDS_DSTBLENDCAPS

Match the **dwDestBlendCaps** member of the **D3DPRIMCAPS** structure specified as the **dpcPrimCaps** member of this structure.

D3DFDS_GUID

Match the globally unique identifier (GUID) specified in the **guid** member of this structure.

D3DFDS_HARDWARE

Match the hardware or software search specification given in the **bHardware** member of this structure.

D3DFDS_LINES

Match the **D3DPRIMCAPS** structure specified by the **dpcLineCaps** member of the **D3DDEVICEDESC** structure.

D3DFDS_MISCCAPS

Match the **dwMiscCaps** member of the **D3DPRIMCAPS** structure specified as the **dpcPrimCaps** member of this structure.

D3DFDS_RASTERCAPS

Match the **dwRasterCaps** member of the **D3DPRIMCAPS** structure

specified as the **dpcPrimCaps** member of this structure.

D3DFDS_SHADECAPS

Match the **dwShadeCaps** member of the **D3DPRIMCAPS** structure specified as the **dpcPrimCaps** member of this structure.

D3DFDS_SRCBLENDCAPS

Match the **dwSrcBlendCaps** member of the **D3DPRIMCAPS** structure specified as the **dpcPrimCaps** member of this structure.

D3DFDS_TEXTUREBLENDCAPS

Match the **dwTextureBlendCaps** member of the **D3DPRIMCAPS** structure specified as the **dpcPrimCaps** member of this structure.

D3DFDS_TEXTURECAPS

Match the **dwTextureCaps** member of the **D3DPRIMCAPS** structure specified as the **dpcPrimCaps** member of this structure.

D3DFDS_TEXTUREFILTERCAPS

Match the **dwTextureFilterCaps** member of the **D3DPRIMCAPS** structure specified as the **dpcPrimCaps** member of this structure.

D3DFDS_TRIANGLES

Match the **D3DPRIMCAPS** structure specified by the **dpcTriCaps** member of the **D3DDEVICEDESC** structure.

D3DFDS_ZCMPCAPS

Match the **dwZCmpCaps** member of the **D3DPRIMCAPS** structure specified as the **dpcPrimCaps** member of this structure.

bHardware

Flag specifying whether the device to find is implemented as hardware or software. If this member is TRUE, the device to search for has hardware rasterization and may also provide other hardware acceleration. Applications that use this flag should set the D3DFDS_HARDWARE bit in the **dwFlags** member.

dcmColorModel

One of the members of the **D3DCOLORMODEL** enumerated type, specifying whether the device to find should use the ramp or RGB color model.

guid

Globally unique identifier (GUID) of the device to find.

dwCaps

Capability flags.

dpcPrimCaps

Specifies a **D3DPRIMCAPS** structure defining the device's capabilities for each primitive type.

See also **D3DFINDDEVICERESULT**

D3DHVERTEX

```
typedef struct _D3DHVERTEX {
    DWORD        dwFlags;
    union {
        D3DVALUE hx;
        D3DVALUE dvHX;
    };
    union {
        D3DVALUE hy;
        D3DVALUE dvHY;
    };
    union {
        D3DVALUE hz;
        D3DVALUE dvHZ;
    };
} D3DHVERTEX, *LPD3DHVERTEX;
```

Defines a homogeneous vertex used when the application is supplying screen coordinate data that needs clipping. This structure is part of the **D3DTRANSFORMDATA** structure.

dwFlags
Flags defining the clip status of the homogeneous vertex. This member can be one or more of the flags described in the **dwClip** member of the **D3DTRANSFORMDATA** structure.

dvHX, **dvHY**, and **dvHZ**
Values of the **D3DVALUE** type describing transformed homogeneous coordinates. These coordinates define the vertex.

D3DINSTRUCTION

```
typedef struct _D3DINSTRUCTION {
    BYTE bOpcode;
    BYTE bSize;
    WORD wCount;
} D3DINSTRUCTION, *LPD3DINSTRUCTION;
```

Defines an instruction in an execute buffer. A display list is made up from a list of variable length instructions. Each instruction begins with a common instruction header and is followed by the data required for that instruction.

bOpcode
Rendering operation, specified as a member of the **D3DOPCODE** enumerated type.

bSize
Size of each instruction data unit. This member can be used to skip to the next instruction in the sequence.

wCount
Number of data units of instructions that follow. This member allows efficient processing of large batches of similar instructions, such as triangles that make up a triangle mesh.

D3DLIGHT

```
typedef struct _D3DLIGHT {
    DWORD         dwSize;
    D3DLIGHTTYPE  dltType;
    D3DCOLORVALUE dcvColor;
    D3DVECTOR     dvPosition;
    D3DVECTOR     dvDirection;
    D3DVALUE      dvRange;
    D3DVALUE      dvFalloff;
    D3DVALUE      dvAttenuation0;
    D3DVALUE      dvAttenuation1;
    D3DVALUE      dvAttenuation2;
    D3DVALUE      dvTheta;
    D3DVALUE      dvPhi;
} D3DLIGHT, *LPD3DLIGHT;
```

Defines the light type in calls to methods such as **IDirect3DLight::SetLight** and **IDirect3DLight::GetLight**.

dwSize
Size, in bytes, of this structure.

dltType
Type of the light source. This value is one of the members of the **D3DLIGHTTYPE** enumerated type.

dcvColor
Color of the light. This member is a **D3DCOLORVALUE** structure.

dvPosition and **dvDirection**
Position and direction of the light in world space.

dvRange
Distance beyond which the light has no effect.

dvFalloff
Decrease in illumination between the umbra (the angle specified by the **dvTheta** member) and the outer edge of the penumbra (the angle specified by the **dvPhi** member).

dvAttenuation0
Constant light intensity. Specifies a light level that does not decrease between the light and the cutoff point given by the **dvRange** member.

dvAttenuation1
Light intensity that decreases linearly. The light intensity is 50 percent of this value halfway between the light and the cutoff point given by the **dvRange** member.

dvAttenuation2
Light intensity that decreases according to a quadratic attenuation factor.

dvTheta
Angle, in radians, of the spotlight's umbra—that is, the fully illuminated spotlight cone.

dvPhi
Angle, in radians, defining the outer edge of the spotlight's penumbra. Points outside this cone are not lit by the spotlight.

The system uses all three of the attenuation settings to determine how the effect of a light decreases with distance from the source. The following equation shows how the attentuation settings are interpreted. The value *d* here is the distance between the vertex being lit and the light:

$$attenuation = attenuation_0 + attenuation_1 \times d + attenuation_2 \times d^2$$

For more information about lights, see **Lighting Module**.

See also **D3DLIGHTTYPE**

D3DLIGHTDATA

```
typedef struct _D3DLIGHTDATA {
    DWORD                dwSize;
    LPD3DLIGHTINGELEMENT lpIn;
    DWORD                dwInSize;
    LPD3DTLVERTEX        lpOut;
    DWORD                dwOutSize;
} D3DLIGHTDATA, *LPD3DLIGHTDATA;
```

Describes the points to be lit and resulting colors in calls to the **IDirect3DViewport::LightElements** method.

dwSize
Size, in bytes, of this structure.

lpIn
Address of a **D3DLIGHTINGELEMENT** structure specifying the input positions and normal vectors.

dwInSize
Amount to skip from one input element to the next. This allows the application to store extra data inline with the element.

lpOut
Address of a **D3DTLVERTEX** structure specifying the output colors.

dwOutSize
Amount to skip from one output color to the next. This allows the application to store extra data inline with the color.

D3DLIGHTINGCAPS

```
typedef struct _D3DLIGHTINGCAPS {
    DWORD dwSize;
    DWORD dwCaps;
    DWORD dwLightingModel;
    DWORD dwNumLights;
} D3DLIGHTINGCAPS, *LPD3DLIGHTINGCAPS;
```

Describes the lighting capabilities of a device. This structure is a member of the **D3DDEVICEDESC** structure.

dwSize
Size, in bytes, of this structure.

dwCaps
Flags describing the capabilities of the lighting module. The following flags are defined:

D3DLIGHTCAPS_DIRECTIONAL
Directional lights are supported.

D3DLIGHTCAPS_GLSPOT
OpenGL-style spotlights are supported.

D3DLIGHTCAPS_PARALLELPOINT
Parallel point lights are supported.

D3DLIGHTCAPS_POINT
Point lights are supported.

D3DLIGHTCAPS_SPOT
Spotlights are supported.

dwLightingModel
Flags defining whether the lighting model is RGB or monochrome. The following flags are defined:

D3DLIGHTINGMODEL_MONO	Monochromatic lighting model.
D3DLIGHTINGMODEL_RGB	RGB lighting model.

dwNumLights
Number of lights that can be handled.

D3DLIGHTINGELEMENT

```
typedef struct _D3DLIGHTINGELEMENT {
    D3DVECTOR dvPosition;
    D3DVECTOR dvNormal;
} D3DLIGHTINGELEMENT, *LPD3DLIGHTINGELEMENT;
```

Describes the points in model space that will be lit. This structure is part of the **D3DLIGHTDATA** structure.

dvPosition
Value specifying the lightable point in model space. This value is a **D3DVECTOR** structure.

dvNormal
Value specifying the normalized unit vector. This value is a **D3DVECTOR** structure.

See also **D3DLIGHTDATA**, **IDirect3DViewport::LightElements**

D3DLINE

```
typedef struct _D3DLINE {
    union {
        WORD v1;
        WORD wV1;
    };
    union {
        WORD v2;
        WORD wV2;
    };
} D3DLINE, *LPD3DLINE;
```

Describes a line for the **D3DOP_LINE** opcode in the **D3DOPCODE** enumerated type.

wV1 and **wV2**
Vertex indices.

Because lines are rendered by using a list of vertices that are to be joined, one less than the count of lines will be rendered.

D3DLINEPATTERN

```
typedef struct _D3DLINEPATTERN {
    WORD wRepeatFactor;
    WORD wLinePattern;
} D3DLINEPATTERN;
```

Describes a line pattern. These values are used by the **D3DRENDERSTATE_LINEPATTERN** render state in the **D3DRENDERSTATETYPE** enumerated type.

wRepeatFactor
Number of bits in the pattern specified by the **wLinePattern** member to use before starting over again at the beginning of the pattern.

wLinePattern
Bits specifying the line pattern. For example, the following value would produce a dotted line: 1100110011001100.

D3DLVERTEX

```
typedef struct _D3DLVERTEX {
    union {
        D3DVALUE x;
        D3DVALUE dvX;
    };
    union {
        D3DVALUE y;
        D3DVALUE dvY;
    };
    union {
        D3DVALUE z;
        D3DVALUE dvZ;
    };
    DWORD        dwReserved;
    union {
        D3DCOLOR color;
        D3DCOLOR dcColor;
    };
    union {
        D3DCOLOR specular;
        D3DCOLOR dcSpecular;
    };
    union {
        D3DVALUE tu;
        D3DVALUE dvTU;
    };
    union {
        D3DVALUE tv;
        D3DVALUE dvTV;
    };
} D3DLVERTEX, *LPD3DLVERTEX;
```

Defines an untransformed and lit vertex (model coordinates with color). Applications should use this structure if the hardware handles vertex transformations. This structure contains only data and a color that would be filled by software lighting.

dvX, **dvY**, and **dvZ**
Values of the **D3DVALUE** type specifying the homogeneous coordinates of the vertex.

dwReserved
Reserved; must be zero.

dcColor and **dcSpecular**
Values of the **D3DCOLOR** type specifying the color and specular component of the vertex.

dvTU and **dvTV**
Values of the **D3DVALUE** type specifying the texture coordinates of the vertex.

D3DMATERIAL

```
typedef struct _D3DMATERIAL {
    DWORD             dwSize;
    union {
        D3DCOLORVALUE diffuse;
        D3DCOLORVALUE dcvDiffuse;
    };
    union {
        D3DCOLORVALUE ambient;
        D3DCOLORVALUE dcvAmbient;
    };
    union {
        D3DCOLORVALUE specular;
        D3DCOLORVALUE dcvSpecular;
    };
    union {
        D3DCOLORVALUE emissive;
        D3DCOLORVALUE dcvEmissive;
    };
    union {
        D3DVALUE      power;
        D3DVALUE      dvPower;
    };
    D3DTEXTUREHANDLE  hTexture;
    DWORD             dwRampSize;
} D3DMATERIAL, *LPD3DMATERIAL;
```

Specifies material properties in calls to the **IDirect3DMaterial::GetMaterial** and **IDirect3DMaterial::SetMaterial** methods.

dwSize
Size, in bytes, of this structure.

dcvDiffuse, **dcvAmbient**, **dcvSpecular**, and **dcvEmissive**
Values specifying the diffuse color, ambient color, specular color, and emissive color of the material, respectively. These values are **D3DCOLORVALUE** structures.

dvPower
Value of the **D3DVALUE** type specifying the sharpness of specular highlights.

hTexture
Handle of the texture map.

dwRampSize
Size of the color ramp. For the monochromatic (ramp) driver, this value must be less than or equal to 1 for materials assigned to the background; otherwise, the background is not displayed. This behavior also occurs when a texture that is assigned to the background has an associated material whose **dwRampSize** member is greater than 1.

The texture handle is acquired from Direct3D by loading a texture into the device. The texture handle may be used only when it has been loaded into the device.

See also **IDirect3DMaterial::GetMaterial**, **IDirect3DMaterial::SetMaterial**

D3DMATRIX

```
typedef struct _D3DMATRIX {
    D3DVALUE _11, _12, _13, _14;
    D3DVALUE _21, _22, _23, _24;
    D3DVALUE _31, _32, _33, _34;
    D3DVALUE _41, _42, _43, _44;
} D3DMATRIX, *LPD3DMATRIX;
```

Describes a matrix for such methods as **IDirect3DDevice::GetMatrix** and **IDirect3DDevice::SetMatrix**.

See also **IDirect3DDevice::GetMatrix**,**IDirect3DDevice::SetMatrix**

D3DMATRIXLOAD

```
typedef struct _D3DMATRIXLOAD {
    D3DMATRIXHANDLE hDestMatrix;
    D3DMATRIXHANDLE hSrcMatrix;
} D3DMATRIXLOAD, *LPD3DMATRIXLOAD;
```

Describes the operand data for the **D3DOP_MATRIXLOAD** opcode in the **D3DOPCODE** enumerated type.

hDestMatrix and **hSrcMatrix**
Handles of the destination and source matrices. These values are **D3DMATRIX** structures.

See also **D3DOPCODE**

D3DMATRIXMULTIPLY

```
typedef struct _D3DMATRIXMULTIPLY {
    D3DMATRIXHANDLE hDestMatrix;
    D3DMATRIXHANDLE hSrcMatrix1;
    D3DMATRIXHANDLE hSrcMatrix2;
} D3DMATRIXMULTIPLY, *LPD3DMATRIXMULTIPLY;
```

Describes the operand data for the **D3DOP_MATRIXMULTIPLY** opcode in the **D3DOPCODE** enumerated type.

hDestMatrix
Handle of the matrix that stores the product of the source matrices. This value is a **D3DMATRIX** structure.

hSrcMatrix1 and **hSrcMatrix2**
Handles of the first and second source matrices. These values are **D3DMATRIX** structures.

See also **D3DOPCODE**

D3DPICKRECORD

```
typedef struct _D3DPICKRECORD {
    BYTE     bOpcode;
    BYTE     bPad;
    DWORD    dwOffset;
    D3DVALUE dvZ;
} D3DPICKRECORD, *LPD3DPICKRECORD;
```

Returns information about picked primitives in an execute buffer for the **IDirect3DDevice::GetPickRecords** method.

bOpcode
Opcode of the picked primitive.

bPad
Pad byte.

dwOffset
Offset from the start of the execute buffer in which the picked primitive was found.

dvZ
Depth of the picked primitive.

The x- and y-coordinates of the picked primitive are specified in the call to the **IDirect3DDevice::Pick** method that created the pick records.

See also **IDirect3DDevice::GetPickRecords**, **IDirect3DDevice::Pick**

D3DPOINT

```
typedef struct _D3DPOINT {
    WORD wCount;
    WORD wFirst;
} D3DPOINT, *LPD3DPOINT;
```

Describes operand data for the **D3DOP_POINT** opcode in the in **D3DOPCODE** enumerated type.

wCount
Number of points.

wFirst
Index of the first vertex.

Points are rendered by using a list of vertices.

See also **D3DOPCODE**

D3DPRIMCAPS

```
typedef struct _D3DPrimCaps {
    DWORD dwSize;
    DWORD dwMiscCaps;
    DWORD dwRasterCaps;
    DWORD dwZCmpCaps;
    DWORD dwSrcBlendCaps;
    DWORD dwDestBlendCaps;
    DWORD dwAlphaCmpCaps;
    DWORD dwShadeCaps;
    DWORD dwTextureCaps;
    DWORD dwTextureFilterCaps;
    DWORD dwTextureBlendCaps;
    DWORD dwTextureAddressCaps;
    DWORD dwStippleWidth;
    DWORD dwStippleHeight;
} D3DPRIMCAPS, *LPD3DPRIMCAPS;
```

Defines the capabilities for each primitive type. This structure is used when creating a device and when querying the capabilities of a device. This structure defines several members in the **D3DDEVICEDESC** structure.

dwSize
Size, in bytes, of this structure.

dwMiscCaps
General capabilities for this primitive. This member can be one or more of the following:

D3DPMISCCAPS_CONFORMANT

The device conforms to the OpenGL standard.

D3DPMISCCAPS_CULLCCW

The driver supports counterclockwise culling through the D3DRENDERSTATE_CULLMODE state. (This applies only to triangle primitives.) This corresponds to the **D3DCULL_CCW** member of the **D3DCULL** enumerated type.

D3DPMISCCAPS_CULLCW

The driver supports clockwise triangle culling through the D3DRENDERSTATE_CULLMODE state. (This applies only to triangle primitives.) This corresponds to the **D3DCULL_CW** member of the **D3DCULL** enumerated type.

D3DPMISCCAPS_CULLNONE

The driver does not perform triangle culling. This corresponds to the **D3DCULL_NONE** member of the **D3DCULL** enumerated type.

D3DPMISCCAPS_LINEPATTERNREP

The driver can handle values other than 1 in the **wRepeatFactor** member of the **D3DLINEPATTERN** structure. (This applies only to line-drawing primitives.)

D3DPMISCCAPS_MASKPLANES

The device can perform a bitmask of color planes.

D3DPMISCCAPS_MASKZ

The device can enable and disable modification of the z-buffer on pixel operations.

dwRasterCaps

Information on raster-drawing capabilities. This member can be one or more of the following:

D3DPRASTERCAPS_DITHER

The device can dither to improve color resolution.

D3DPRASTERCAPS_FOGTABLE

The device calculates the fog value by referring to a lookup table containing fog values that are indexed to the depth of a given pixel.

D3DPRASTERCAPS_FOGVERTEX

The device calculates the fog value during the lighting operation, places the value into the alpha component of the **D3DCOLOR** value given for the **specular** member of the **D3DTLVERTEX** structure, and interpolates the fog value during rasterization.

D3DPRASTERCAPS_PAT

The driver can perform patterned drawing (lines or fills with D3DRENDERSTATE_LINEPATTERN or one of the D3DRENDERSTATE_STIPPLEPATTERN render states) for the

primitive being queried.

D3DPRASTERCAPS_ROP2

The device can support raster operations other than R2_COPYPEN.

D3DPRASTERCAPS_STIPPLE

The device can stipple polygons to simulate translucency.

D3DPRASTERCAPS_SUBPIXEL

The device performs subpixel placement of z, color, and texture data, rather than working with the nearest integer pixel coordinate. This helps avoid bleed-through due to z imprecision, and jitter of color and texture values for pixels. Note that there is no corresponding state that can be enabled and disabled; the device either performs subpixel placement or it does not, and this bit is present only so that the Direct3D client will be better able to determine what the rendering quality will be.

D3DPRASTERCAPS_SUBPIXELX

The device is subpixel accurate along the x-axis only and is clamped to an integer y-axis scanline. For information about subpixel accuracy, see D3DPRASTERCAPS_SUBPIXEL.

D3DPRASTERCAPS_XOR

The device can support **XOR** operations. If this flag is not set but D3DPRIM_RASTER_ROP2 is set, then **XOR** operations must still be supported.

D3DPRASTERCAPS_ZTEST

The device can perform z-test operations. This effectively renders a primitive and indicates whether any z pixels would have been rendered.

dwZCmpCaps

Z-buffer comparison functions that the driver can perform. This member can be one or more of the following:

D3DPCMPCAPS_ALWAYS

Always pass the z test.

D3DPCMPCAPS_EQUAL

Pass the z test if the new z equals the current z.

D3DPCMPCAPS_GREATER

Pass the z test if the new z is greater than the current z.

D3DPCMPCAPS_GREATEREQUAL

Pass the z test if the new z is greater than or equal to the current z.

D3DPCMPCAPS_LESS

Pass the z test if the new z is less than the current z.

D3DPCMPCAPS_LESSEQUAL

Pass the z test if the new z is less than or equal to the current z.

D3DPCMPCAPS_NEVER

Always fail the z test.

D3DPCMPCAPS_NOTEQUAL

Pass the z test if the new z does not equal the current z.

dwSrcBlendCaps

Source blending capabilities. This member can be one or more of the following. (The RGBA values of the source and destination are indicated with the subscripts *s* and *d*.)

D3DPBLENDCAPS_BOTHINVSRCALPHA

Source blend factor is (1-As, 1-As, 1-As, 1-As) and destination blend factor is (As, As, As, As); the destination blend selection is overridden.

D3DPBLENDCAPS_BOTHSRCALPHA

Source blend factor is (As, As, As, As) and destination blend factor is (1-As, 1-As, 1-As, 1-As); the destination blend selection is overridden.

D3DPBLENDCAPS_DESTALPHA

Blend factor is (Ad, Ad, Ad, Ad).

D3DPBLENDCAPS_DESTCOLOR

Blend factor is (Rd, Gd, Bd, Ad).

D3DPBLENDCAPS_INVDESTALPHA

Blend factor is (1-Ad, 1-Ad, 1-Ad, 1-Ad).

D3DPBLENDCAPS_INVDESTCOLOR

Blend factor is (1-Rd, 1-Gd, 1-Bd, 1-Ad).

D3DPBLENDCAPS_INVSRCALPHA

Blend factor is (1-As, 1-As, 1-As, 1-As).

D3DPBLENDCAPS_INVSRCCOLOR

Blend factor is (1-Rd, 1-Gd, 1-Bd, 1-Ad).

D3DPBLENDCAPS_ONE

Blend factor is (1, 1, 1, 1).

D3DPBLENDCAPS_SRCALPHA

Blend factor is (As, As, As, As).

D3DPBLENDCAPS_SRCALPHASAT

Blend factor is (f, f, f, 1); f = min(As, 1-Ad).

D3DPBLENDCAPS_SRCCOLOR

Blend factor is (Rs, Gs, Bs, As).

D3DPBLENDCAPS_ZERO

Blend factor is (0, 0, 0, 0).

dwDestBlendCaps

Destination blending capabilities. This member can be the same capabilities that are defined for the **dwSrcBlendCaps** member.

dwAlphaCmpCaps

Alpha-test comparison functions that the driver can perform. This member can be the same capabilities that are defined for the **dwZCmpCaps** member.

dwShadeCaps

Shading operations that the device can perform. It is assumed, in general, that if a device supports a given command (such as **D3DOP_TRIANGLE**) at all, it supports the D3DSHADE_FLAT mode (as specified in the **D3DSHADEMODE** enumerated type). This flag specifies whether the driver can also support Gouraud and Phong shading and whether alpha color components are supported for each of the three color-generation modes. When alpha components are not supported in a given mode, the alpha value of colors generated in that mode is implicitly 255. This is the maximum possible alpha (that is, the alpha component is at full intensity).

The color, specular highlights, fog, and alpha interpolants of a triangle each have capability flags that an application can use to find out how they are implemented by the device driver. These are modified by the shade mode, color model, and by whether the alpha component of a color is blended or stippled. For more information, see **Polygons**.

This member can be one or more of the following:

D3DPSHADECAPS_ALPHAFLATBLEND

D3DPSHADECAPS_ALPHAFLATSTIPPLED

Device can support an alpha component for flat blended and stippled transparency, respectively (the D3DSHADE_FLAT state for the **D3DSHADEMODE** enumerated type). In these modes, the alpha color component for a primitive is provided as part of the color for the first vertex of the primitive.

D3DPSHADECAPS_ALPHAGOURAUDBLEND

D3DPSHADECAPS_ALPHAGOURAUDSTIPPLED

Device can support an alpha component for Gouraud blended and stippled transparency, respectively (the D3DSHADE_GOURAUD state for the **D3DSHADEMODE** enumerated type). In these modes, the alpha color component for a primitive is provided at vertices and interpolated across a face along with the other color components.

D3DPSHADECAPS_ALPHAPHONGBLEND

D3DPSHADECAPS_ALPHAPHONGSTIPPLED

Device can support an alpha component for Phong blended and stippled transparency, respectively (the D3DSHADE_PHONG state for the **D3DSHADEMODE** enumerated type). In these modes, vertex parameters are reevaluated on a per-pixel basis, applying lighting effects for the red, green, and blue color components. Phong shading is not supported for DirectX 2.

D3DPSHADECAPS_COLORFLATMONO

D3DPSHADECAPS_COLORFLATRGB

Device can support colored flat shading in the **D3DCOLOR_MONO** and **D3DCOLOR_RGB** color models, respectively. In these modes, the color component for a primitive is provided as part of the color for the first vertex of the primitive. In monochromatic lighting modes, only the blue component of the color is interpolated; in RGB lighting modes, of course, the red, green, and blue components are interpolated.

D3DPSHADECAPS_COLORGOURAUDMONO

D3DPSHADECAPS_COLORGOURAUDRGB

Device can support colored Gouraud shading in the **D3DCOLOR_MONO** and **D3DCOLOR_RGB** color models, respectively. In these modes, the color component for a primitive is provided at vertices and interpolated across a face along with the other color components. In monochromatic lighting modes, only the blue component of the color is interpolated; in RGB lighting modes, of course, the red, green, and blue components are interpolated.

D3DPSHADECAPS_COLORPHONGMONO

D3DPSHADECAPS_COLORPHONGRGB

Device can support colored Phong shading in the **D3DCOLOR_MONO** and **D3DCOLOR_RGB** color models, respectively. In these modes, vertex parameters are reevaluated on a per-pixel basis. Lighting effects are applied for the red, green, and blue color components in RGB mode, and for the blue component only for monochromatic mode. Phong shading is not supported for DirectX 2.

D3DPSHADECAPS_FOGFLAT

D3DPSHADECAPS_FOGGOURAUD

D3DPSHADECAPS_FOGPHONG

Device can support fog in the flat, Gouraud, and Phong shading models, respectively. Phong shading is not supported for DirectX 2.

D3DPSHADECAPS_SPECULARFLATMONO

D3DPSHADECAPS_SPECULARFLATRGB

Device can support specular highlights in flat shading in the **D3DCOLOR_MONO** and **D3DCOLOR_RGB** color models, respectively.

D3DPSHADECAPS_SPECULARGOURAUDMONO

D3DPSHADECAPS_SPECULARGOURAUDRGB

Device can support specular highlights in Gouraud shading in the **D3DCOLOR_MONO** and **D3DCOLOR_RGB** color models, respectively.

D3DPSHADECAPS_SPECULARPHONGMONO

D3DPSHADECAPS_SPECULARPHONGRGB

Device can support specular highlights in Phong shading in the **D3DCOLOR_MONO** and **D3DCOLOR_RGB** color models, respectively. Phong shading is not supported for DirectX 2.

dwTextureCaps

Miscellaneous texture-mapping capabilities. This member can be one or more of the following:

D3DPTEXTURECAPS_ALPHA

RGBA textures are supported in the D3DTEX_DECAL and D3DTEX_MODULATE texture filtering modes. If this capability is not set, then only RGB textures are supported in those modes. Regardless of the setting of this flag, alpha must always be supported in D3DTEX_DECAL_MASK, D3DTEX_DECAL_ALPHA, and D3DTEX_MODULATE_ALPHA filtering modes whenever those filtering modes are available.

D3DPTEXTURECAPS_BORDER

Texture mapping along borders is supported.

D3DPTEXTURECAPS_PERSPECTIVE

Perspective correction is supported.

D3DPTEXTURECAPS_POW2

All non-mipmapped textures must have widths and heights specified as powers of two if this flag is set. (Note that all mipmapped textures must always have dimensions that are powers of two.)

D3DPTEXTURECAPS_SQUAREONLY

All textures must be square.

D3DPTEXTURECAPS_TRANSPARENCY

Texture transparency is supported. (Only those texels that are not the current transparent color are drawn.)

dwTextureFilterCaps

Texture-mapping capabilities. This member can be one or more of the following:

D3DPTFILTERCAPS_LINEAR

A weighted average of a 2-by-2 area of texels surrounding the desired pixel is used. This applies to both zooming in and zooming out. If either zooming in or zooming out is supported, then both must be supported.

D3DPTFILTERCAPS_LINEARMIPLINEAR

Similar to D3DPRIM_TEX_MIP_LINEAR, but interpolates between the two nearest mipmaps.

D3DPTFILTERCAPS_LINEARMIPNEAREST

Similar to D3DPRIM_TEX_MIP_NEAREST, but interpolates between the two nearest mipmaps.

D3DPTFILTERCAPS_MIPLINEAR

Similar to D3DPRIM_TEX_LINEAR, but uses the appropriate mipmap for texel selection.

D3DPTFILTERCAPS_MIPNEAREST

Similar to D3DPRIM_TEX_NEAREST, but uses the appropriate mipmap for texel selection.

D3DPTFILTERCAPS_NEAREST

The texel with coordinates nearest to the desired pixel value is used. This applies to both zooming in and zooming out. If either zooming in or zooming out is supported, then both must be supported.

dwTextureBlendCaps

Texture-blending capabilities. See the **D3DTEXTUREBLEND** enumerated type for discussions of the various texture-blending modes. This member can be one or more of the following:

D3DPTBLENDCAPS_COPY

Copy mode texture-blending (**D3DTBLEND_COPY** from the **D3DTEXTUREBLEND** enumerated type) is supported.

D3DPTBLENDCAPS_DECAL

Decal texture-blending mode (**D3DTBLEND_DECAL** from the **D3DTEXTUREBLEND** enumerated type) is supported.

D3DPTBLENDCAPS_DECALALPHA

Decal-alpha texture-blending mode (**D3DTBLEND_DECALALPHA** from the **D3DTEXTUREBLEND** enumerated type) is supported.

D3DPTBLENDCAPS_DECALMASK

Decal-mask texture-blending mode (**D3DTBLEND_DECALMASK** from the **D3DTEXTUREBLEND** enumerated type) is supported.

D3DPTBLENDCAPS_MODULATE

Modulate texture-blending mode (**D3DTBLEND_MODULATE** from the **D3DTEXTUREBLEND** enumerated type) is supported.

D3DPTBLENDCAPS_MODULATEALPHA

Modulate-alpha texture-blending mode (**D3DTBLEND_MODULATEALPHA** from the **D3DTEXTUREBLEND** enumerated type) is supported.

D3DPTBLENDCAPS_MODULATEMASK

Modulate-mask texture-blending mode (**D3DTBLEND_MODULATEMASK** from the **D3DTEXTUREBLEND** enumerated type) is supported.

dwTextureAddressCaps

Texture-addressing capabilities. This member can be one or more of the following:

D3DPTADDRESSCAPS_CLAMP

Device can clamp textures to addresses.

D3DPTADDRESSCAPS_MIRROR

Device can mirror textures to addresses.

D3DPTADDRESSCAPS_WRAP

Device can wrap textures to addresses.

dwStippleWidth and **dwStippleHeight**

Maximum width and height of the supported stipple (up to 32-by-32).

D3DPROCESSVERTICES

```
typedef struct _D3DPROCESSVERTICES {
    DWORD dwFlags;
    WORD  wStart;
    WORD  wDest;
    DWORD dwCount;
    DWORD dwReserved;
} D3DPROCESSVERTICES, *LPD3DPROCESSVERTICES;
```

Describes how vertices in the execute buffer should be handled by the driver. This is used by the **D3DOP_PROCESSVERTICES** opcode in the **D3DOPCODE** enumerated type.

dwFlags

One or more of the following flags indicating how the driver should process the vertices:

D3DPROCESSVERTICES_COPY

Vertices should simply be copied to the driver, because they have always been transformed and lit. If all the vertices in the execute buffer can be copied, the driver does not need to do the work of processing the vertices, and a performance improvement results.

D3DPROCESSVERTICES_NOCOLOR

Vertices should not be colored.

D3DPROCESSVERTICES_OPMASK

Specifies a bitmask of the other flags in the **dwFlags** member, exclusive of **D3DPROCESSVERTICES_NOCOLOR** and **D3DPROCESSVERTICES_UPDATEEXTENTS**.

D3DPROCESSVERTICES_TRANSFORM

Vertices should be transformed.

D3DPROCESSVERTICES_TRANSFORMLIGHT

Vertices should be transformed and lit.

D3DPROCESSVERTICES_UPDATEEXTENTS

Extents of all transformed vertices should be updated. This information is

returned in the **drExtent** member of the **D3DSTATUS** structure.

wStart
Index of the first vertex in the source.

wDest
Index of the first vertex in the local buffer.

dwCount
Number of vertices to be processed.

dwReserved
Reserved; must be zero.

See also **D3DOPCODE**

D3DRECT

```
typedef struct _D3DRECT {
    union {
        LONG x1;
        LONG lX1;
    };
    union {
        LONG y1;
        LONG lY1;
    };
    union {
        LONG x2;
        LONG lX2;
    };
    union {
        LONG y2;
        LONG lY2;
    };
} D3DRECT, *LPD3DRECT;
;
```

Rectangle definition.

lX1 and **lY1**
Coordinates of the upper-left corner of the rectangle.

lX2 and **lY2**
Coordinates of the lower-right corner of the rectangle.

See also **D3DRMUPDATECALLBACK**, **IDirect3DDevice::Pick**, **IDirect3DViewport::Clear**

D3DSPAN

```
typedef struct _D3DSPAN {
```

```
    WORD wCount;
    WORD wFirst;
} D3DSPAN, *LPD3DSPAN;
```

Defines a span for the **D3DOP_SPAN** opcode in the **D3DOPCODE** enumerated type. Spans join a list of points with the same y value. If the y value changes, a new span is started.

wCount
Number of spans.

wFirst
Index to first vertex.

See also **D3DOPCODE**

D3DSTATE

```
typedef struct _D3DSTATE {
    union {
        D3DTRANSFORMSTATETYPE dtstTransformStateType;
        D3DLIGHTSTATETYPE     dlstLightStateType;
        D3DRENDERSTATETYPE    drstRenderStateType;
    };
    union {
        DWORD                 dwArg[1];
        D3DVALUE              dvArg[1];
    };
} D3DSTATE, *LPD3DSTATE;
```

Describes the render state for the **D3DOP_STATETRANSFORM**, **D3DOP_STATELIGHT**, and **D3DOP_STATERENDER** opcodes in the **D3DOPCODE** enumerated type. The first member of this structure is the relevant enumerated type and the second is the value for that type.

dtstTransformStateType, **dlstLightStateType**, and **drstRenderStateType**
One of the members of the **D3DTRANSFORMSTATETYPE**, **D3DLIGHTSTATETYPE**, or **D3DRENDERSTATETYPE** enumerated type specifying the render state.

dvArg
Value of the type specified in the first member of this structure.

See also **D3DLIGHTSTATETYPE**, **D3DOPCODE**, **D3DRENDERSTATETYPE**, and **D3DTRANSFORMSTATETYPE**, **D3DVALUE**

D3DSTATS

```
typedef struct _D3DSTATS {
```

```
    DWORD dwSize;
    DWORD dwTrianglesDrawn;
    DWORD dwLinesDrawn;
    DWORD dwPointsDrawn;
    DWORD dwSpansDrawn;
    DWORD dwVerticesProcessed;
} D3DSTATS, *LPD3DSTATS;
```

Contains statistics used by the **IDirect3DDevice::GetStats** method.

dwSize
Size, in bytes, of this structure.

dwTrianglesDrawn, **dwLinesDrawn**, **dwPointsDrawn**, and **dwSpansDrawn**
Number of triangles, lines, points, and spans drawn since the device was created.

dwVerticesProcessed
Number of vertices processed since the device was created.

See also **IDirect3DDevice::GetStats**

D3DSTATUS

```
typedef struct _D3DSTATUS {
    DWORD   dwFlags;
    DWORD   dwStatus;
    D3DRECT drExtent;
} D3DSTATUS, *LPD3DSTATUS;
```

Describes the current status of the execute buffer. This structure is part of the **D3DEXECUTEDATA** structure and is used with the **D3DOP_SETSTATUS** opcode in the **D3DOPCODE** enumerated type.

dwFlags
One of the following flags, specifying whether the status, the extents, or both are being set:

D3DSETSTATUS_STATUS
Set the status.

D3DSETSTATUS_EXTENTS
Set the extents specified in the **drExtent** member.

D3DSETSTATUS_ALL
Set both the status and the extents.

dwStatus
Clipping flags. This member can be one or more of the following flags:

Combination and General Flags

D3DSTATUS_CLIPINTERSECTION

Combination of all CLIPINTERSECTION flags.

D3DSTATUS_CLIPUNIONALL

Combination of all CLIPUNION flags.

D3DSTATUS_DEFAULT

Combination of D3DSTATUS_CLIPINTERSECTION and D3DSTATUS_ZNOTVISIBLE flags. This value is the default.

D3DSTATUS_ZNOTVISIBLE

Clip Intersection Flags

D3DSTATUS_CLIPINTERSECTIONLEFT

Logical **AND** of the clip flags for the vertices compared to the left side of the viewing frustum.

D3DSTATUS_CLIPINTERSECTIONRIGHT

Logical **AND** of the clip flags for the vertices compared to the right side of the viewing frustum.

D3DSTATUS_CLIPINTERSECTIONTOP

Logical **AND** of the clip flags for the vertices compared to the top of the viewing frustum.

D3DSTATUS_CLIPINTERSECTIONBOTTOM

Logical **AND** of the clip flags for the vertices compared to the bottom of the viewing frustum.

D3DSTATUS_CLIPINTERSECTIONFRONT

Logical **AND** of the clip flags for the vertices compared to the front clipping plane of the viewing frustum.

D3DSTATUS_CLIPINTERSECTIONBACK

Logical **AND** of the clip flags for the vertices compared to the back clipping plane of the viewing frustum.

D3DSTATUS_CLIPINTERSECTIONGEN0 through
D3DSTATUS_CLIPINTERSECTIONGEN5

Logical **AND** of the clip flags for application-defined clipping planes.

Clip Union Flags

D3DSTATUS_CLIPUNIONLEFT

Equal to D3DCLIP_LEFT.

D3DSTATUS_CLIPUNIONRIGHT

Equal to D3DCLIP_RIGHT.

D3DSTATUS_CLIPUNIONTOP

Equal to D3DCLIP_TOP.

D3DSTATUS_CLIPUNIONBOTTOM

Equal to D3DCLIP_BOTTOM.

D3DSTATUS_CLIPUNIONFRONT

Equal to D3DCLIP_FRONT.

D3DSTATUS_CLIPUNIONBACK

Equal to D3DCLIP_BACK.

D3DSTATUS_CLIPUNIONGEN0 through **D3DSTATUS_CLIPUNIONGEN5**

Equal to D3DCLIP_GEN0 through D3DCLIP_GEN5.

Basic Clipping Flags

D3DCLIP_BACK

All vertices are clipped by the back plane of the viewing frustum.

D3DCLIP_BOTTOM

All vertices are clipped by the bottom plane of the viewing frustum.

D3DCLIP_FRONT

All vertices are clipped by the front plane of the viewing frustum.

D3DCLIP_LEFT

All vertices are clipped by the left plane of the viewing frustum.

D3DCLIP_RIGHT

All vertices are clipped by the right plane of the viewing frustum.

D3DCLIP_TOP

All vertices are clipped by the top plane of the viewing frustum.

D3DCLIP_GEN0 through **D3DCLIP_GEN5**

Application-defined clipping planes.

drExtent

A **D3DRECT** structure that defines a bounding box for all the relevant vertices. For example, the structure might define the area containing the output of the **D3DOP_PROCESSVERTICES** opcode, assuming the D3DPROCESSVERTICES_UPDATEEXTENTS flag is set in the **D3DPROCESSVERTICES** structure.

The status is a rolling status and is updated during each execution. The bounding box in the **drExtent** member can grow with each execution, but it does not shrink; it can be reset only by using the **D3DOP_SETSTATUS** opcode.

See also **D3DEXECUTEDATA**, **D3DOPCODE**, **D3DRECT**

D3DTEXTURELOAD

```
typedef struct _D3DTEXTURELOAD {
    D3DTEXTUREHANDLE hDestTexture;
    D3DTEXTUREHANDLE hSrcTexture;
} D3DTEXTURELOAD, *LPD3DTEXTURELOAD;
```

Describes operand data for the **D3DOP_TEXTURELOAD** opcode in the **D3DOPCODE** enumerated type.

hDestTexture
Handle of the destination texture.

hSrcTexture
Handle of the source texture.

The textures referred to by the **hDestTexture** and **hSrcTexture** members must be the same size.

D3DTLVERTEX

```
typedef struct _D3DTLVERTEX {
    union {
        D3DVALUE sx;
        D3DVALUE dvSX;
    };
    union {
        D3DVALUE sy;
        D3DVALUE dvSY;
    };
    union {
        D3DVALUE sz;
        D3DVALUE dvSZ;
    };
    union {
        D3DVALUE rhw;
        D3DVALUE dvRHW;
    };
    union {
        D3DCOLOR color;
        D3DCOLOR dcColor;
    };
    union {
        D3DCOLOR specular;
        D3DCOLOR dcSpecular;
    };
    union {
        D3DVALUE tu;
        D3DVALUE dvTU;
    };
    union {
        D3DVALUE tv;
        D3DVALUE dvTV;
    };
} D3DTLVERTEX, *LPD3DTLVERTEX;
```

Defines a transformed and lit vertex (screen coordinates with color) for the **D3DLIGHTDATA** structure.

dvSX, **dvSY**, and **dvSZ**
Values of the **D3DVALUE** type describing a vertex in screen coordinates.

dvRHW
Value of the **D3DVALUE** type that is the reciprocal of homogeneous w. This value is 1 divided by the distance from the origin to the object along the z-axis.

dcColor and **dcSpecular**
Values of the **D3DCOLOR** type describing the color and specular component of the vertex.

dvTU and **dvTV**
Values of the **D3DVALUE** type describing the texture coordinates of the vertex.

See also **D3DLIGHTDATA**

D3DTRANSFORMCAPS

```
typedef struct _D3DTransformCaps {
    DWORD dwSize;
    DWORD dwCaps;
} D3DTRANSFORMCAPS, *LPD3DTRANSFORMCAPS;
```

Describes the transformation capabilities of a device. This structure is part of the **D3DDEVICEDESC** structure.

dwSize
Size, in bytes, of this structure.

dwCaps
Flag specifying whether the system clips while transforming. This member can be zero or the following flag:

D3DTRANSFORMCAPS_CLIP The system clips while transforming.

D3DTRANSFORMDATA

```
typedef struct _D3DTRANSFORMDATA {
    DWORD        dwSize;
    LPVOID       lpIn;
    DWORD        dwInSize;
    LPVOID       lpOut;
    DWORD        dwOutSize;
    LPD3DHVERTEX lpHOut;
    DWORD        dwClip;
    DWORD        dwClipIntersection;
    DWORD        dwClipUnion;
    D3DRECT      drExtent;
} D3DTRANSFORMDATA, *LPD3DTRANSFORMDATA;
```

Contains information about transformations for the **IDirect3DViewport::TransformVertices** method.

dwSize
Size of the structure, in bytes.

lpIn
Address of the vertices to be transformed.

dwInSize
Stride of the vertices to be transformed.

lpOut
Address used to store the transformed vertices.

dwOutSize
Stride of output vertices.

lpHOut
Address of a value that contains homogeneous transformed vertices. This value is a **D3DHVERTEX** structure

dwClip
Flags specifying how the vertices are clipped. This member can be one or more of the following values:

D3DCLIP_BACK
Clipped by the back plane of the viewing frustum.

D3DCLIP_BOTTOM
Clipped by the bottom plane of the viewing frustum.

D3DCLIP_FRONT
Clipped by the front plane of the viewing frustum.

D3DCLIP_LEFT
Clipped by the left plane of the viewing frustum.

D3DCLIP_RIGHT
Clipped by the right plane of the viewing frustum.

D3DCLIP_TOP
Clipped by the top plane of the viewing frustum.

D3DCLIP_GEN0 through **D3DCLIP_GEN5**
Application-defined clipping planes.

dwClipIntersection
Flags denoting the intersection of the clip flags. This member can be one or more of the following values:

D3DSTATUS_CLIPINTERSECTIONLEFT
Logical **AND** of the clip flags for the vertices compared to the left side of

the viewing frustum.

D3DSTATUS_CLIPINTERSECTIONRIGHT

Logical **AND** of the clip flags for the vertices compared to the right side of the viewing frustum.

D3DSTATUS_CLIPINTERSECTIONTOP

Logical **AND** of the clip flags for the vertices compared to the top of the viewing frustum.

D3DSTATUS_CLIPINTERSECTIONBOTTOM

Logical **AND** of the clip flags for the vertices compared to the bottom of the viewing frustum.

D3DSTATUS_CLIPINTERSECTIONFRONT

Logical **AND** of the clip flags for the vertices compared to the front clipping plane of the viewing frustum.

D3DSTATUS_CLIPINTERSECTIONBACK

Logical **AND** of the clip flags for the vertices compared to the back clipping plane of the viewing frustum.

D3DSTATUS_CLIPINTERSECTIONGEN0 through **D3DSTATUS_CLIPINTERSECTIONGEN5**

Logical **AND** of the clip flags for application-defined clipping planes.

dwClipUnion

Flags denoting the union of the clip flags. This member can be one or more of the following values:

D3DSTATUS_CLIPUNIONLEFT

Equal to D3DCLIP_LEFT.

D3DSTATUS_CLIPUNIONRIGHT

Equal to D3DCLIP_RIGHT.

D3DSTATUS_CLIPUNIONTOP

Equal to D3DCLIP_TOP.

D3DSTATUS_CLIPUNIONBOTTOM

Equal to D3DCLIP_BOTTOM.

D3DSTATUS_CLIPUNIONFRONT

Equal to D3DCLIP_FRONT.

D3DSTATUS_CLIPUNIONBACK

Equal to D3DCLIP_BACK.

D3DSTATUS_CLIPUNIONGEN0 through **D3DSTATUS_CLIPUNIONGEN5**

Equal to D3DCLIP_GEN0 through D3DCLIP_GEN5.

drExtent
Value that defines the extent of the transformed vertices. This structure is filled by the transformation module with the screen extent of the transformed geometry. For geometries that are clipped, this extent will only include vertices that are inside the viewing volume. This value is a **D3DRECT** structure

Each input vertex should be a three-vector vertex giving the [x y z] coordinates in model space for the geometry. The **dwInSize** member gives the amount to skip between vertices, allowing the application to store extra data inline with each vertex.

All values generated by the transformation module are stored as 16-bit precision values. The clip is treated as an integer bitfield that is set to the inclusive **OR** of the viewing volume planes that clip a given transformed vertex.

See also **IDirect3DViewport::TransformVertices**

D3DTRIANGLE

```
typedef struct _D3DTRIANGLE {
    union {
        WORD v1;
        WORD wV1;
    };
    union {
        WORD v2;
        WORD wV2;
    };
    union {
        WORD v3;
        WORD wV3;
    };
    WORD     wFlags;
} D3DTRIANGLE, *LPD3DTRIANGLE;
```

Describes the base type for all triangles. The triangle is the main rendering primitive.

For related information, see the **D3DOP_TRIANGLE** member in the **D3DOPCODE** enumerated type.

wV1, **wV2**, and **wV3**
Vertices describing the triangle.

wFlags
Flags describing which edges of the triangle to enable. (This information is useful only in wireframe mode.) This value can be a combination of the following edge and strip and fan flags:

Edge flags

D3DTRIFLAG_EDGEENABLE1

Edge defined by **v1**–**v2**.

D3DTRIFLAG_EDGEENABLE2

Edge defined by **v2**–**v3**.

D3DTRIFLAG_EDGEENABLE3

Edge defined by **v3**–**v1**.

D3DTRIFLAG_EDGEENABLETRIANGLE

All edges.

Strip and fan flags

D3DTRIFLAG_EVEN

The **v1**–**v2** edge of the current triangle is adjacent to the **v3**–**v1** edge of the previous triangle; that is, **v1** is the previous **v1**, and **v2** is the previous **v3**.

D3DTRIFLAG_ODD

The **v1**–**v2** edge of the current triangle is adjacent to the **v2**–**v3** edge of the previous triangle; that is, **v1** is the previous **v3**, and **v2** is the previous **v2**.

D3DTRIFLAG_START

Begin the strip or fan, loading all three vertices.

D3DTRIFLAG_STARTFLAT(len)

If this triangle is culled, also cull the specified number of subsequent triangles. This length must be greater than zero and less than 30.

This structure can be used directly for all triangle fills. For flat shading, the color and specular components are taken from the first vertex. The three vertex indices **v1**, **v2**, and **v3** are vertex indexes into the vertex list at the start of the execute buffer.

Enabled edges are visible in wireframe mode. When an application displays wireframe triangles that share an edge, it typically enables only one (or neither) edge to avoid drawing the edge twice.

The D3DTRIFLAG_ODD and D3DTRIFLAG_EVEN flags refer to the locations of a triangle in a conventional triangle strip or fan. If a triangle strip had five triangles, the following flags would be used to define the strip:

D3DTRIFLAG_START

D3DTRIFLAG_ODD

D3DTRIFLAG_EVEN

D3DTRIFLAG_ODD

D3DTRIFLAG_EVEN

Similarly, the following flags would define a triangle fan with five triangles:

D3DTRIFLAG_START
D3DTRIFLAG_EVEN
D3DTRIFLAG_EVEN
D3DTRIFLAG_EVEN
D3DTRIFLAG_EVEN

The following flags could define a flat triangle fan with five triangles:

D3DTRIFLAG_STARTFLAT(4)
D3DTRIFLAG_EVEN
D3DTRIFLAG_EVEN
D3DTRIFLAG_EVEN
D3DTRIFLAG_EVEN

See also **Triangle Strips and Fans**

D3DVECTOR

```
typedef struct _D3DVECTOR {
    union {
        D3DVALUE x;
        D3DVALUE dvX;
    };
    union {
        D3DVALUE y;
        D3DVALUE dvY;
    };
    union {
        D3DVALUE z;
        D3DVALUE dvZ;
    };
} D3DVECTOR, *LPD3DVECTOR;
```

Defines a vector for many Direct3D and Direct3DRM methods and structures.

dvX, **dvY**, and **dvZ**
Values of the **D3DVALUE** type describing the vector.

See also **D3DLIGHT**, **D3DLIGHTINGELEMENT**, **D3DRMBOX**, **D3DRMQUATERNION**, **D3DRMVERTEX**

D3DVERTEX

```
typedef struct _D3DVERTEX {
    union {
```

```
        D3DVALUE x;
        D3DVALUE dvX;
    };
    union {
        D3DVALUE y;
        D3DVALUE dvY;
    };
    union {
        D3DVALUE z;
        D3DVALUE dvZ;
    };
    union {
        D3DVALUE nx;
        D3DVALUE dvNX;
    };
    union {
        D3DVALUE ny;
        D3DVALUE dvNY;
    };
    union {
        D3DVALUE nz;
        D3DVALUE dvNZ;
    };
    union {
        D3DVALUE tu;
        D3DVALUE dvTU;
    };
    union {
        D3DVALUE tv;
        D3DVALUE dvTV;
    };
} D3DVERTEX, *LPD3DVERTEX;
```

Defines an untransformed and unlit vertex (model coordinates with normal direction vector).

For related information, see the **D3DOP_TRIANGLE** member in the **D3DOPCODE** enumerated type.

dvX, **dvY**, and **dvZ**
Values of the **D3DVALUE** type describing the homogeneous coordinates of the vertex.

dvNX, **dvNY**, and **dvNZ**
Values of the **D3DVALUE** type describing the normal coordinates of the vertex.

dvTU and **dvTV**
Values of the **D3DVALUE** type describing the texture coordinates of the vertex.

See also **D3DVALUE**

D3DVIEWPORT

```
typedef struct _D3DVIEWPORT {
    DWORD    dwSize;
    DWORD    dwX;
    DWORD    dwY;
    DWORD    dwWidth;
    DWORD    dwHeight;
    D3DVALUE dvScaleX;
    D3DVALUE dvScaleY;
    D3DVALUE dvMaxX;
    D3DVALUE dvMaxY;
    D3DVALUE dvMinZ;
    D3DVALUE dvMaxZ;
} D3DVIEWPORT, *LPD3DVIEWPORT;
```

Defines the visible 3D volume and the 2D screen area that a 3D volume projects onto for the **IDirect3DViewport::GetViewport** and **IDirect3DViewport::SetViewport** methods.

When the viewport is changed, the driver builds a new transformation matrix.

The coordinates and dimensions of the viewport are given relative to the top left of the device.

dwSize
Size of this structure, in bytes.

dwX and **dwY**
Coordinates of the top-left corner of the viewport.

dwWidth and **dwHeight**
Dimensions of the viewport.

dvScaleX and **dvScaleY**
Values of the **D3DVALUE** type describing the scaling quantities homogeneous to screen.

dvMaxX, **dvMaxY**, **dvMinZ**, and **dvMaxZ**
Values of the **D3DVALUE** type describing the maximum and minimum homogeneous coordinates of x, y, and z.

See also **D3DVALUE**, **IDirect3DViewport::GetViewport**, **IDirect3DViewport::SetViewport**

Enumerated Types

D3DBLEND

```
typedef enum _D3DBLEND {
    D3DBLEND_ZERO            = 1,
    D3DBLEND_ONE             = 2,
```

```
    D3DBLEND_SRCCOLOR        = 3,
    D3DBLEND_INVSRCCOLOR     = 4,
    D3DBLEND_SRCALPHA        = 5,
    D3DBLEND_INVSRCALPHA     = 6,
    D3DBLEND_DESTALPHA       = 7,
    D3DBLEND_INVDESTALPHA    = 8,
    D3DBLEND_DESTCOLOR       = 9,
    D3DBLEND_INVDESTCOLOR    = 10,
    D3DBLEND_SRCALPHASAT     = 11,
    D3DBLEND_BOTHSRCALPHA    = 12,
    D3DBLEND_BOTHINVSRCALPHA = 13,
} D3DBLEND;
```

Defines the supported blend modes for the **D3DRENDERSTATE_DSTBLEND** values in the **D3DRENDERSTATETYPE** enumerated type. In the member descriptions that follow, the RGBA values of the source and destination are indicated with the subscripts *s* and *d*.

D3DBLEND_ZERO
Blend factor is (0, 0, 0, 0).

D3DBLEND_ONE
Blend factor is (1, 1, 1, 1).

D3DBLEND_SRCCOLOR
Blend factor is (R_s, G_s, B_s, A_s).

D3DBLEND_INVSRCCOLOR
Blend factor is (A_s, A_s, A_s, A_s, 1-A_s).

D3DBLEND_SRCALPHA
Blend factor is (A_s, A_s, A_s, A_s).

D3DBLEND_INVSRCALPHA
Blend factor is (1-A_s, 1-A_s, 1-A_s).

D3DBLEND_DESTALPHA
Blend factor is (A_d, A_d, A_d, A_d).

D3DBLEND_INVDESTALPHA
Blend factor is (1-A_d, 1-A_d, 1-A_d, 1-A_d).

D3DBLEND_DESTCOLOR
Blend factor is (R_d, G_d, B_d, A_d).

D3DBLEND_INVDESTCOLOR
Blend factor is (1-R_d, 1-G_d, 1-B_d, 1-A_d).

D3DBLEND_SRCALPHASAT
Blend factor is (f, f, f, 1); f = min(A_s, 1-A_d).

D3DBLEND_BOTHSRCALPHA
Source blend factor is (A_s, A_s, A_s, A_s), and destination blend factor is (1-A_s, 1-A_s, 1-A_s, 1-A_s); the destination blend selection is overridden.

D3DBLEND_BOTHINVSRCALPHA
Source blend factor is (1-As, 1-As, 1-As, 1-As), and destination blend factor is (As, As, As, As); the destination blend selection is overridden.

D3DCMPFUNC

```
typedef enum _D3DCMPFUNC {
    D3DCMP_NEVER        = 1,
    D3DCMP_LESS         = 2,
    D3DCMP_EQUAL        = 3,
    D3DCMP_LESSEQUAL    = 4,
    D3DCMP_GREATER      = 5,
    D3DCMP_NOTEQUAL     = 6,
    D3DCMP_GREATEREQUAL = 7,
    D3DCMP_ALWAYS       = 8,
} D3DCMPFUNC;
```

Defines the supported compare functions for the **D3DRENDERSTATE_ZFUNC** and **D3DRENDERSTATE_ALPHAFUNC** values of the **D3DRENDERSTATETYPE** enumerated type.

D3DCMP_NEVER
Always fail the test.

D3DCMP_LESS
Accept the new pixel if its value is less than the value of the current pixel.

D3DCMP_EQUAL
Accept the new pixel if its value equals the value of the current pixel.

D3DCMP_LESSEQUAL
Accept the new pixel if its value is less than or equal to the value of the current pixel.

D3DCMP_GREATER
Accept the new pixel if its value is greater than the value of the current pixel.

D3DCMP_NOTEQUAL
Accept the new pixel if its value does not equal the value of the current pixel.

D3DCMP_GREATEREQUAL
Accept the new pixel if its value is greater than or equal to the value of the current pixel.

D3DCMP_ALWAYS
Always pass the test.

D3DCOLORMODEL

```
typedef enum _D3DCOLORMODEL {
    D3DCOLOR_MONO = 1,
    D3DCOLOR_RGB  = 2,
} D3DCOLORMODEL;
```

Defines the color model that the system will run in.

D3DCOLOR_MONO
Use a monochromatic model (or ramp model). In this model, the blue component of a vertex color is used to define the brightness of a lit vertex.

D3DCOLOR_RGB
Use a full RGB model.

See also **D3DDEVICEDESC**, **D3DFINDDEVICESEARCH**, **D3DLIGHTSTATETYPE**, **IDirect3DRMDevice::GetColorModel**

D3DCULL

```
typedef enum _D3DCULL {
    D3DCULL_NONE = 1,
    D3DCULL_CW   = 2,
    D3DCULL_CCW  = 3,
} D3DCULL;
```

Defines the supported cull modes. These define how faces are culled when rendering a geometry.

D3DCULL_NONE
Do not cull faces.

D3DCULL_CW
Cull faces with clockwise vertices.

D3DCULL_CCW
Cull faces with counterclockwise vertices.

See also **D3DPRIMCAPS**, **D3DRENDERSTATETYPE**

D3DFILLMODE

```
typedef enum _D3DFILLMODE {
    D3DFILL_POINT     = 1,
    D3DFILL_WIREFRAME = 2,
    D3DFILL_SOLID     = 3
} D3DFILLMODE;
```

Contains constants describing the fill mode. These values are used by the **D3DRENDERSTATE_FILLMODE** render state in the **D3DRENDERSTATETYPE** enumerated type.

D3DFILL_POINT
Fill points.

D3DFILL_WIREFRAME
Fill wireframes.

D3DFILL_SOLID
Fill solids.

D3DFOGMODE

```
typedef enum _D3DFOGMODE {
    D3DFOG_NONE   = 0,
    D3DFOG_EXP    = 1,
    D3DFOG_EXP2   = 2,
    D3DFOG_LINEAR = 3
} D3DFOGMODE;
```

Contains constants describing the fog mode. These values are used by the **D3DRENDERSTATE_FOGTABLEMODE** render state in the **D3DRENDERSTATETYPE** enumerated type.

D3DFOG_NONE
No fog effect.

D3DFOG_EXP
The fog effect intensifies exponentially, according to the following formula:

$$f = e^{-(density \times z)}$$

D3DFOG_EXP2
The fog effect intensifies exponentially with the square of the distance, according to the following formula:

$$f = e^{-(density \times z)^2}$$

D3DFOG_LINEAR
The fog effect intensifies linearly between the start and end points, according to the following formula:

$$f = \frac{end - z}{end - start}$$

This is the only fog mode supported for DirectX 2.

Note that fog can be considered a measure of visibility—the lower the fog value produced by one of the fog equations, the less visible an object is.

D3DLIGHTSTATETYPE

```
typedef enum _D3DLIGHTSTATETYPE {
    D3DLIGHTSTATE_MATERIAL   = 1,
```

```
    D3DLIGHTSTATE_AMBIENT    = 2,
    D3DLIGHTSTATE_COLORMODEL = 3,
    D3DLIGHTSTATE_FOGMODE    = 4,
    D3DLIGHTSTATE_FOGSTART   = 5,
    D3DLIGHTSTATE_FOGEND     = 6,
    D3DLIGHTSTATE_FOGDENSITY = 7,
} D3DLIGHTSTATETYPE;
```

Defines the light state for the **D3DOP_STATELIGHT** opcode. This enumerated type is part of the **D3DSTATE** structure.

D3DLIGHTSTATE_MATERIAL
Defines the material associated with a light. The default value is NULL.

D3DLIGHTSTATE_AMBIENT
Defines whether the light is ambient. The default value is 0.

D3DLIGHTSTATE_COLORMODEL
One of the members of the **D3DCOLORMODEL** enumerated type. The default value is D3DCOLOR_RGB.

D3DLIGHTSTATE_FOGMODE
One of the members of the **D3DFOGMODE** enumerated type. The default value is D3DFOG_NONE.

D3DLIGHTSTATE_FOGSTART
Defines the starting value for fog. The default value is 1.0.

D3DLIGHTSTATE_FOGEND
Defines the ending value for fog. The default value is 100.0.

D3DLIGHTSTATE_FOGDENSITY
Defines the density setting for fog. The default value is 1.0.

See also **D3DOPCODE** and **D3DSTATE**

D3DLIGHTTYPE

```
typedef enum _D3DLIGHTTYPE {
    D3DLIGHT_POINT         = 1,
    D3DLIGHT_SPOT          = 2,
    D3DLIGHT_DIRECTIONAL   = 3,
    D3DLIGHT_PARALLELPOINT = 4,
    D3DLIGHT_GLSPOT        = 5,
} D3DLIGHTTYPE;
```

Defines the light type. This enumerated type is part of the **D3DLIGHT** structure.

D3DLIGHT_POINT
Light is a point source.

D3DLIGHT_SPOT
Light is a spotlight source.

D3DLIGHT_DIRECTIONAL
Light is a directional source.

D3DLIGHT_PARALLELPOINT
Light is a parallel source.

D3DLIGHT_GLSPOT
Light is a GL-style spotlight.

See also **Direct3DRMLight**

D3DOPCODE

```
typedef enum _D3DOPCODE {
    D3DOP_POINT           = 1,
    D3DOP_LINE            = 2,
    D3DOP_TRIANGLE        = 3,
    D3DOP_MATRIXLOAD      = 4,
    D3DOP_MATRIXMULTIPLY  = 5,
    D3DOP_STATETRANSFORM  = 6,
    D3DOP_STATELIGHT      = 7,
    D3DOP_STATERENDER     = 8,
    D3DOP_PROCESSVERTICES = 9,
    D3DOP_TEXTURELOAD     = 10,
    D3DOP_EXIT            = 11,
    D3DOP_BRANCHFORWARD   = 12,
    D3DOP_SPAN            = 13,
    D3DOP_SETSTATUS       = 14,
} D3DOPCODE;
```

Contains the opcodes for execute buffers.

D3DOP_POINT
Sends a point to the renderer. Operand data is described by the **D3DPOINT** structure.

D3DOP_LINE
Sends a line to the renderer. Operand data is described by the **D3DLINE** structure.

D3DOP_TRIANGLE
Sends a triangle to the renderer. Operand data is described by the **D3DTRIANGLE** structure.

D3DOP_MATRIXLOAD
Triggers a data transfer in the rendering engine. Operand data is described by the **D3DMATRIXLOAD** structure.

D3DOP_MATRIXMULTIPLY
Triggers a data transfer in the rendering engine. Operand data is described by the **D3DMATRIXMULTIPLY** structure.

D3DOP_STATETRANSFORM
Sets the value of internal state variables in the rendering engine for the transformation module. Operand data is a variable token and the new value. The token identifies the internal state variable, and the new value is the value to which that variable should be set. For more information about these variables, see the **D3DSTATE** structure and the **D3DTRANSFORMSTATETYPE** enumerated type.

D3DOP_STATELIGHT
Sets the value of internal state variables in the rendering engine for the lighting module. Operand data is a variable token and the new value. The token identifies the internal state variable, and the new value is the value to which that variable should be set. For more information about these variables, see the **D3DSTATE** structure and the **D3DLIGHTSTATETYPE** enumerated type.

D3DOP_STATERENDER
Sets the value of internal state variables in the rendering engine for the rendering module. Operand data is a variable token and the new value. The token identifies the internal state variable, and the new value is the value to which that variable should be set. For more information about these variables, see the **D3DSTATE** structure and the **D3DRENDERSTATETYPE** enumerated type.

D3DOP_PROCESSVERTICES
Sets both lighting and transformations for vertices. Operand data is described by the **D3DPROCESSVERTICES** structure.

D3DOP_TEXTURELOAD
Triggers a data transfer in the rendering engine. Operand data is described by the **D3DTEXTURELOAD** structure.

D3DOP_EXIT
Signals that the end of the list has been reached.

D3DOP_BRANCHFORWARD
Enables a branching mechanism within the execute buffer. For more information, see the **D3DBRANCH** structure.

D3DOP_SPAN
Spans a list of points with the same y value. For more information, see the **D3DSPAN** structure.

D3DOP_SETSTATUS
Resets the status of the execute buffer. For more information, see the **D3DSTATUS** structure.

An execute buffer has two parts: an array of vertices (each typically with position, normal vector, and texture coordinates) and an array of opcode/operand groups. One opcode can have several operands following it; the system simply performs the relevant operation on each operand.

See also **D3DINSTRUCTION**

D3DRENDERSTATETYPE

```
typedef enum _D3DRENDERSTATETYPE {
    D3DRENDERSTATE_TEXTUREHANDLE      = 1,
    D3DRENDERSTATE_ANTIALIAS          = 2,
    D3DRENDERSTATE_TEXTUREADDRESS     = 3,
    D3DRENDERSTATE_TEXTUREPERSPECTIVE = 4,
    D3DRENDERSTATE_WRAPU              = 5,
    D3DRENDERSTATE_WRAPV              = 6,
    D3DRENDERSTATE_ZENABLE            = 7,
    D3DRENDERSTATE_FILLMODE           = 8,
    D3DRENDERSTATE_SHADEMODE          = 9,
    D3DRENDERSTATE_LINEPATTERN        = 10,
    D3DRENDERSTATE_MONOENABLE         = 11,
    D3DRENDERSTATE_ROP2               = 12,
    D3DRENDERSTATE_PLANEMASK          = 13,
    D3DRENDERSTATE_ZWRITEENABLE       = 14,
    D3DRENDERSTATE_ALPHATESTENABLE    = 15,
    D3DRENDERSTATE_LASTPIXEL          = 16,
    D3DRENDERSTATE_TEXTUREMAG         = 17,
    D3DRENDERSTATE_TEXTUREMIN         = 18,
    D3DRENDERSTATE_SRCBLEND           = 19,
    D3DRENDERSTATE_DESTBLEND          = 20,
    D3DRENDERSTATE_TEXTUREMAPBLEND    = 21,
    D3DRENDERSTATE_CULLMODE           = 22,
    D3DRENDERSTATE_ZFUNC              = 23,
    D3DRENDERSTATE_ALPHAREF           = 24,
    D3DRENDERSTATE_ALPHAFUNC          = 25,
    D3DRENDERSTATE_DITHERENABLE       = 26,
    D3DRENDERSTATE_BLENDENABLE        = 27,
    D3DRENDERSTATE_FOGENABLE          = 28,
    D3DRENDERSTATE_SPECULARENABLE     = 29,
    D3DRENDERSTATE_ZVISIBLE           = 30,
    D3DRENDERSTATE_SUBPIXEL           = 31,
    D3DRENDERSTATE_SUBPIXELX          = 32,
    D3DRENDERSTATE_STIPPLEDALPHA      = 33,
    D3DRENDERSTATE_FOGCOLOR           = 34,
    D3DRENDERSTATE_FOGTABLEMODE       = 35,
    D3DRENDERSTATE_FOGTABLESTART      = 36,
    D3DRENDERSTATE_FOGTABLEEND        = 37,
    D3DRENDERSTATE_FOGTABLEDENSITY    = 38,
    D3DRENDERSTATE_STIPPLEENABLE      = 39,
    D3DRENDERSTATE_STIPPLEPATTERN00   = 64,
       // Stipple patterns 01 through 30 omitted here.
    D3DRENDERSTATE_STIPPLEPATTERN31   = 95,
} D3DRENDERSTATETYPE;
```

Describes the render state for the **D3DOP_STATERENDER** opcode. This enumerated type is part of the **D3DSTATE** structure. The values mentioned in the following descriptions are set in the second member of this structure.

D3DRENDERSTATE_TEXTUREHANDLE
Texture handle. The default value is NULL.

D3DRENDERSTATE_ANTIALIAS
Antialiasing primitive edges. The default value is FALSE.

D3DRENDERSTATE_TEXTUREADDRESS
One of the members of the **D3DTEXTUREADDRESS** enumerated type. The default value is D3DTADDRESS_WRAP.

D3DRENDERSTATE_TEXTUREPERSPECTIVE
TRUE for perspective correction. The default value is FALSE.

D3DRENDERSTATE_WRAPU
TRUE for wrapping in u direction. The default value is FALSE.

D3DRENDERSTATE_WRAPV
TRUE for wrapping in v direction. The default value is FALSE.

D3DRENDERSTATE_ZENABLE
TRUE to enable the z-buffer comparison test when writing to the frame buffer. The default value is FALSE.

D3DRENDERSTATE_FILLMODE
One or more members of the **D3DFILLMODE** enumerated type. The default value is D3DFILL_SOLID.

D3DRENDERSTATE_SHADEMODE
One or more members of the **D3DSHADEMODE** enumerated type. The default value is D3DSHADE_GOURAUD.

D3DRENDERSTATE_LINEPATTERN
The **D3DLINEPATTERN** structure. The default values are 0 for **wRepeatPattern** and 0 for **wLinePattern**.

D3DRENDERSTATE_ROP2
One of the 16 ROP2 binary raster operations specifying how the supplied pixels are combined with the pixels of the display surface. The default value is R2_COPYPEN. Applications can use the **D3DPRASTERCAPS_ROP2** flag in the **dwRasterCaps** member of the **D3DPRIMCAPS** structure to determine whether additional raster operations are supported.

D3DRENDERSTATE_PLANEMASK
Physical plane mask whose type is **ULONG**. The default value is ~0.

D3DRENDERSTATE_ZWRITEENABLE
TRUE to enable z writes. The default value is TRUE. This member enables an application to prevent the system from updating the z-buffer with new z values.

D3DRENDERSTATE_ALPHATESTENABLE
TRUE to enable alpha tests. The default value is FALSE. This member enables applications to turn off the tests that otherwise would accept or reject a pixel based on its alpha value.

D3DRENDERSTATE_LASTPIXEL
TRUE to prevent drawing the last pixel in a line. The default value is FALSE.

D3DRENDERSTATE_TEXTUREMAG
One of the members of the **D3DTEXTUREFILTER** enumerated type. The default value is D3DFILTER_NEAREST.

D3DRENDERSTATE_TEXTUREMIN
One of the members of the **D3DTEXTUREFILTER** enumerated type. The default value is D3DFILTER_NEAREST.

D3DRENDERSTATE_SRCBLEND
One of the members of the **D3DBLEND** enumerated type. The default value is D3DBLEND_ONE.

D3DRENDERSTATE_DSTBLEND
One of the members of the **D3DBLEND** enumerated type. The default value is D3DBLEND_ZERO.

D3DRENDERSTATE_TEXTUREMAPBLEND
One of the members of the **D3DTEXTUREBLEND** enumerated type. The default value is D3DTBLEND_MODULATE.

D3DRENDERSTATE_CULLMODE
One of the members of the **D3DCULL** enumerated type. The default value is D3DCULL_CCW. Software renderers have a fixed culling order and do not support changing the culling mode.

D3DRENDERSTATE_ZFUNC
One of the members of the **D3DCMPFUNC** enumerated type. The default value is D3DCMP_LESSEQUAL. This member enables an application to accept or reject a pixel based on its distance from the camera.

D3DRENDERSTATE_ALPHAREF
Value specifying a reference alpha value against which pixels are tested when alpha-testing is enabled. This value's type is **D3DFIXED**. The default value is 0.

D3DRENDERSTATE_ALPHAFUNC
One of the members of the **D3DCMPFUNC** enumerated type. The default value is D3DCMP_ALWAYS. This member enables an application to accept or reject a pixel based on its alpha value.

D3DRENDERSTATE_DITHERENABLE
TRUE to enable dithering. The default value is FALSE.

D3DRENDERSTATE_BLENDENABLE
TRUE to enable alpha blending. The default value is FALSE.

D3DRENDERSTATE_FOGENABLE
TRUE to enable fog. The default value is FALSE.

D3DRENDERSTATE_SPECULARENABLE
TRUE to enable specular. The default value is TRUE.

D3DRENDERSTATE_ZVISIBLE
TRUE to enable z-checking. The default value is FALSE. Z-checking is a culling technique in which a polygon representing the screen space of an entire group of

polygons is tested against the z-buffer to discover whether any of the polygons should be drawn.

D3DRENDERSTATE_SUBPIXEL
TRUE to enable subpixel correction. The default value is FALSE.

D3DRENDERSTATE_SUBPIXELX
TRUE to enable correction in X only. The default value is FALSE.

D3DRENDERSTATE_STIPPLEDALPHA
TRUE to enable stippled alpha. The default value is FALSE.

D3DRENDERSTATE_FOGCOLOR
Value whose type is **D3DCOLOR**. The default value is 0.

D3DRENDERSTATE_FOGTABLEMODE
One of the members of the **D3DFOGMODE** enumerated type. The default value is D3DFOG_NONE.

D3DRENDERSTATE_FOGTABLESTART
Fog table start. This is the position at which fog effects begin for linear fog mode.

D3DRENDERSTATE_FOGTABLEEND
Fog table end. This is the position at which fog effects reach their maximum density for linear fog mode.

D3DRENDERSTATE_FOGTABLEDENSITY
Sets the maximum fog density for linear fog mode. This value can range from 0 to 1.

D3DRENDERSTATE_STIPPLEENABLE
Enables stippling in the device driver. When stippled alpha is enabled, it must override the current stipple pattern. When stippled alpha is disabled, the stipple pattern must be returned.

D3DRENDERSTATE_STIPPLEPATTERN00 through **D3DRENDERSTATE_STIPPLEPATTERN31**
Stipple pattern. Each render state applies to a separate line of the stipple pattern.

See also **D3DOPCODE**, **D3DSTATE**

D3DSHADEMODE

```
typedef enum _D3DSHADEMODE {
    D3DSHADE_FLAT    = 1,
    D3DSHADE_GOURAUD = 2,
    D3DSHADE_PHONG   = 3,
} D3DSHADEMODE;
```

Describes the supported shade mode for the **D3DRENDERSTATE_SHADEMODE** render state in the **D3DRENDERSTATETYPE** enumerated type.

D3DSHADE_FLAT
Flat shade mode. The color of the first vertex in the triangle is used to determine the color of the face.

D3DSHADE_GOURAUD
Gouraud shade mode. The color of the face is determined by a linear interpolation between all three of the triangle's vertices.

D3DSHADE_PHONG
Phong shade mode is not supported for DirectX 2.

See also **D3DRENDERSTATETYPE**

D3DTEXTUREADDRESS

```
typedef enum _D3DTEXTUREADDRESS {
    D3DTADDRESS_WRAP   = 1,
    D3DTADDRESS_MIRROR = 2,
    D3DTADDRESS_CLAMP  = 3,
} D3DTEXTUREADDRESS;
```

Describes the supported texture address for the **D3DRENDERSTATE_TEXTUREADDRESS** render state in the **D3DRENDERSTATETYPE** enumerated type.

D3DTADDRESS_WRAP
The **D3DRENDERSTATE_WRAPU** and **D3DRENDERSTATE_WRAPV** render states of the **D3DRENDERSTATETYPE** enumerated type are used. This is the default setting.

D3DTADDRESS_MIRROR
Equivalent to a tiling texture-addressing mode (that is, when neither D3DRENDERSTATE_WRAPU nor D3DRENDERSTATE_WRAPV is used) except that the texture is flipped at every integer junction. For u values between 0 and 1, for example, the texture is addressed normally, between 1 and 2 the texture is flipped (mirrored), between 2 and 3 the texture is normal again, and so on.

D3DTADDRESS_CLAMP
Texture coordinates greater than 1.0 are set to 1.0, and values less than 0.0 are set to 0.0.

For more information about using the D3DRENDERSTATE_WRAPU and D3DRENDERSTATE_WRAPV render states, see **Direct3D Texture Objects**.

See also **D3DRENDERSTATETYPE**

D3DTEXTUREBLEND

```
typedef enum _D3DTEXTUREBLEND {
    D3DTBLEND_DECAL         = 1,
    D3DTBLEND_MODULATE      = 2,
```

```
    D3DTBLEND_DECALALPHA    = 3,
    D3DTBLEND_MODULATEALPHA = 4,
    D3DTBLEND_DECALMASK     = 5,
    D3DTBLEND_MODULATEMASK  = 6,
    D3DTBLEND_COPY          = 7,
} D3DTEXTUREBLEND;
```

Defines the supported texture-blending modes. This enumerated type is used by the **D3DRENDERSTATE_TEXTUREMAPBLEND** render state in the **D3DRENDERSTATETYPE** enumerated type.

D3DTBLEND_DECAL
Decal texture-blending mode is supported. In this mode, the RGB and alpha values of the texture replace the colors that would have been used with no texturing.

D3DTBLEND_MODULATE
Modulate texture-blending mode is supported. In this mode, the RGB values of the texture are multiplied with the RGB values that would have been used with no texturing. Any alpha values in the texture replace the alpha values in the colors that would have been used with no texturing.

D3DTBLEND_DECALALPHA
Decal-alpha texture-blending mode is supported. In this mode, the RGB and alpha values of the texture are blended with the colors that would have been used with no texturing, according to the following formula:

$C = (1–A_t)C_o + A_tC_t$

In this formula, C stands for color, A for alpha, $_t$ for texture, and $_o$ for original object (before blending).

In the D3DTBLEND_DECALALPHA mode, any alpha values in the texture replace the alpha values in the colors that would have been used with no texturing.

D3DTBLEND_MODULATEALPHA
Modulate-alpha texture-blending mode is supported. In this mode, the RGB values of the texture are multiplied with the RGB values that would have been used with no texturing, and the alpha values of the texture are multiplied with the alpha values that would have been used with no texturing.

D3DTBLEND_DECALMASK
Decal-mask texture-blending mode is supported.

D3DTBLEND_MODULATEMASK
Modulate-mask texture-blending mode is supported.

D3DTBLEND_COPY
Copy texture-blending mode is supported.

Modulation combines the effects of lighting and texturing. Because colors are specified as values between and including 0 and 1, modulating (multiplying) the texture and pre-existing colors together typically produces colors that are less bright than either source. The brightness of a color component is undiminished

when one of the sources for that component is white (1). The simplest way to ensure that the colors of a texture do not change when the texture is applied to an object is to ensure that the object is white (1,1,1).

D3DTEXTUREFILTER

```
typedef enum _D3DTEXTUREFILTER {
    D3DFILTER_NEAREST          = 1,
    D3DFILTER_LINEAR           = 2,
    D3DFILTER_MIPNEAREST       = 3,
    D3DFILTER_MIPLINEAR        = 4,
    D3DFILTER_LINEARMIPNEAREST = 5,
    D3DFILTER_LINEARMIPLINEAR  = 6,
} D3DTEXTUREFILTER;
```

Defines the supported texture filter modes used by the **D3DRENDERSTATE_TEXTUREMAG** render state in the **D3DRENDERSTATETYPE** enumerated type.

D3DFILTER_NEAREST
The texel with coordinates nearest to the desired pixel value is used. This applies to both zooming in and zooming out. If either zooming in or zooming out is supported, then both must be supported.

D3DFILTER_LINEAR
A weighted average of a 2-by-2 area of texels surrounding the desired pixel is used. This applies to both zooming in and zooming out. If either zooming in or zooming out is supported, then both must be supported.

D3DFILTER_MIPNEAREST
Similar to D3DFILTER_NEAREST, but uses the appropriate mipmap for texel selection.

D3DFILTER_MIPLINEAR
Similar to D3DFILTER_LINEAR, but uses the appropriate mipmap for texel selection.

D3DFILTER_LINEARMIPNEAREST
Similar to D3DFILTER_MIPNEAREST, but interpolates between the two nearest mipmaps.

D3DFILTER_LINEARMIPLINEAR
Similar to D3DFILTER_MIPLINEAR, but interpolates between the two nearest mipmaps.

D3DTRANSFORMSTATETYPE

```
typedef enum _D3DTRANSFORMSTATETYPE {
    D3DTRANSFORMSTATE_WORLD      = 1,
    D3DTRANSFORMSTATE_VIEW       = 2,
    D3DTRANSFORMSTATE_PROJECTION = 3,
```

```
} D3DTRANSFORMSTATETYPE;
```

Describes the transformation state for the **D3DOP_STATETRANSFORM** opcode in the **D3DOPCODE** enumerated type. This enumerated type is part of the **D3DSTATE** structure.

D3DTRANSFORMSTATE_WORLD
D3DTRANSFORMSTATE_VIEW
D3DTRANSFORMSTATE_PROJECTION
Define the matrices for the world, view, and projection transformations. The default values are NULL (the identity matrices).

See also **D3DOPCODE**, **D3DRENDERSTATETYPE**

Other Types

D3DCOLOR

```
typedef DWORD D3DCOLOR, D3DCOLOR, *LPD3DCOLOR;
```

This type is the fundamental Direct3D color type.

See also **D3DRGB**, **D3DRGBA**

D3DVALUE

```
typedef float D3DVALUE, *LPD3DVALUE;
```

This type is the fundamental Direct3D fractional data type.

Return Values

Errors are represented by negative values and cannot be combined. This table lists the values that can be returned by all Direct3D methods. See the individual method descriptions for lists of the values each can return.

D3D_OK

D3DERR_BADMAJORVERSION

D3DERR_BADMINORVERSION

D3DERR_EXECUTE_CLIPPED_FAILED

D3DERR_EXECUTE_CREATE_FAILED

D3DERR_EXECUTE_DESTROY_FAILED

D3DERR_EXECUTE_FAILED

D3DERR_EXECUTE_LOCK_FAILED

D3DERR_EXECUTE_LOCKED

D3DERR_EXECUTE_NOT_LOCKED

D3DERR_EXECUTE_UNLOCK_FAILED

D3DERR_LIGHT_SET_FAILED

D3DERR_MATERIAL_CREATE_FAILED

D3DERR_MATERIAL_DESTROY_FAILED

D3DERR_MATERIAL_GETDATA_FAILED

D3DERR_MATERIAL_SETDATA_FAILED

D3DERR_MATRIX_CREATE_FAILED

D3DERR_MATRIX_DESTROY_FAILED

D3DERR_MATRIX_GETDATA_FAILED

D3DERR_MATRIX_SETDATA_FAILED

D3DERR_SCENE_BEGIN_FAILED

D3DERR_SCENE_END_FAILED

D3DERR_SCENE_IN_SCENE

D3DERR_SCENE_NOT_IN_SCENE

D3DERR_SETVIEWPORTDATA_FAILED

D3DERR_TEXTURE_CREATE_FAILED

D3DERR_TEXTURE_DESTROY_FAILED

D3DERR_TEXTURE_GETSURF_FAILED

D3DERR_TEXTURE_LOAD_FAILED

D3DERR_TEXTURE_LOCK_FAILED

D3DERR_TEXTURE_LOCKED

D3DERR_TEXTURE_NO_SUPPORT

D3DERR_TEXTURE_NOT_LOCKED

D3DERR_TEXTURE_SWAP_FAILED

D3DERR_TEXTURE_UNLOCK_FAILED